Contents

Chapter 1

A Problem in Many Binary Search Implementations

A Problem in Many Binary Search Implementations - GeeksforGeeks

Consider the following C implementation of Binary Search function, is there anything wrong in this?

```
 // A iterative binary search function. It returns location of x in
// given array arr[l..r] if present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r)
    {
        // find index of middle element
        int m = (l+r)/2;

        // Check if x is present at mid
        if (arr[m] == x) return m;

        // If x greater, ignore left half
        if (arr[m] < x) l = m + 1;

        // If x is smaller, ignore right half
        else r = m - 1;
    }

    // if we reach here, then element was not present
    return -1;
}
```

The above looks fine except one subtle thing, the expression "m = (l+r)/2". It fails for large values of l and r. Specifically, it fails if the sum of low and high is greater than the

maximum positive int value ($2^{31} - 1$). The sum overflows to a negative value, and the value stays negative when divided by two. In C this causes an array index out of bounds with unpredictable results.

What is the way to resolve this problem?
Following is one way:

```
int mid = low + ((high - low) / 2);
```

Probably faster, and arguably as clear is (works only in Java, refer this):

```
int mid = (low + high) >>> 1;
```

In C and C++ (where you don't have the >>> operator), you can do this:

```
mid = ((unsigned int)low + (unsigned int)high)) >> 1
```

The similar problem appears in Merge Sort as well.

The above content is taken from google reasearch blog.

Please refer thisas well, it points out that the above solutions may not always work.

The above problem occurs when array length is 2^{30} or greater and the search repeatedly moves to second half of the array. This much size of array is not likely to appear most of the time. For example, when we try the below program with 32 bit Code Blocks compiler, we get compiler error.

```
 int main()
{
    int arr[1<<30];
    return 0;
}
```

Output:

```
error: size of array 'arr' is too large
```

Even when we try boolean array, the program compiles fine, but crashes when run in Windows 7.0 and Code Blocks 32 bit compiler

```
 #include <stdbool.h>
int main()
{
    bool arr[1<<30];
    return 0;
}
```

Output: No compiler error, but crashes at run time.

Sources:
http://googleresearch.blogspot.in/2006/06/extra-extra-read-all-about-it-nearly.html
http://locklessinc.com/articles/binary_search/

This article is contributed by **Abhay Rathi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/problem-binary-search-implementations/

Chapter 2

All unique triplets that sum up to a given value

All unique triplets that sum up to a given value - GeeksforGeeks

Given an array and a sum value, find all possible unique triplets in that array whose sum is equal to the given sum value. If no such triplets can be formed from the array, then print "No triplets can be formed", else print all the unique triplets. For example, if the given array is {12, 3, 6, 1, 6, 9} and given sum is 24, then the unique triplets are (3, 9, 12) and (6, 6, 12) whose sum is 24.

Examples:

```
Input : array = {12, 3, 6, 1, 6, 9} sum = 24
Output : [[3, 9, 12], [6, 6, 12]]

Input : array = {-2, 0, 1, 1, 2} sum = 0
Output : [[-2, 0, 2], [-2, 1, 1]]

Input : array = {-2, 0, 1, 1, 2} sum = 10
Output : No triplets can be formed
```

In a previous post, Find a triplet that sum to a given value we have discussed whether the triplets can be formed from the array or not.

Here we need to print all unique set of triplets that sum up to a given value

1. Sort the input array.
2. Find three indexes from the array i, j and k where A[i]+A[j]+A[k] = given sum value.
3. Fix the first element as A[i] and iterate i from 0 to array size – 2.
4. For each iteration of i, take j to be index of the first element in the remaining elements and k to be index of the last element.
5. Check for the triplet combination A[i]+A[j]+A[k] = given sum value.

6. If triplet is obtained (ie., A[i]+A[j]+A[k] = given sum value)
……6.1. Add all the triplet in a TreeSet with ":" separated value to get the unique triplets.
……6.2. increment the second value index
……6.3. decrement the third value index.
……6.4. repeat step 4 & 5 till j < k
7. If A[i]+A[j]+A[k] given sum value, decrement the third value index

C++

```
 // C++ program to find unique triplets
// that sum up to a given value.
#include <bits/stdc++.h>
using namespace std;

// Structure to define a triplet.
struct triplet{
    int first, second, third;
};

// Function to find unique triplets that
// sum up to a given value.
int findTriplets(int nums[], int n, int sum)
{
    int i, j, k;

    // Vector to store all unique triplets.
    vector <triplet> triplets;

    // Set to store already found triplets
    // to avoid duplication.
    unordered_set <string> uniqTriplets;

    // Variable used to hold triplet
    // converted to string form.
    string temp;

    // Variable used to store current
    // triplet which is stored in vector
    // if it is unique.
    triplet newTriplet;

    // Sort the input array.
    sort(nums, nums + n);

    // Iterate over the array from the
    // start and consider it as the
    // first element.
    for(i = 0; i < n - 2; i++){
```

```
// index of the first element in
// the remaining elements.
j = i + 1;

// index of the last element.
k = n - 1;

while(j < k){

    // If sum of triplet is equal to
    // given value, then check if
    // this triplet is unique or not.
    // To check uniqueness, convert
    // triplet to string form and
    // then check if this string is
    // present in set or not. If
    // triplet is unique, then store
    // it in vector.
    if(nums[i] + nums[j] + nums[k] == sum)
    {
        temp = to_string(nums[i]) + " : "
             + to_string(nums[j]) + " : "
                     + to_string(nums[k]);
        if(uniqTriplets.find(temp) ==
                        uniqTriplets.end())
        {
            uniqTriplets.insert(temp);
            newTriplet.first = nums[i];
            newTriplet.second = nums[j];
            newTriplet.third = nums[k];
            triplets.push_back(newTriplet);
        }

        // Increment the first index
        // and decrement the last
        // index of remaining elements.
        j++;
        k--;
    }

    // If sum is greater than given
    // value then to reduce sum
    // decrement the last index.
    else if(nums[i] + nums[j] +
                        nums[k] > sum)
        k--;

    // If sum is less than given value
```

```
            // then to increase sum increment
            // the first index of remaining
            // elements.
            else
                j++;
        }
    }

    // If no unique triplet is found, then
    // return 0.
    if(triplets.size() == 0)
        return 0;

    // Print all unique triplets stored in
    // vector.
    for(i = 0; i < triplets.size(); i++)
    {
        cout << "[" << triplets[i].first
            << ", " << triplets[i].second
          << ", " << triplets[i].third <<"], ";
    }
}

// Driver Function.
int main()
{
    int nums[] = { 12, 3, 6, 1, 6, 9 };
    int n = sizeof(nums) / sizeof(nums[0]);
    int sum = 24;
    if(!findTriplets(nums, n, sum))
        cout << "No triplets can be formed.";

    return 0;
}

// This code is contributed by NIKHIL JINDAL.
```

Java

```
 // Java program to find all triplets with given sum
import java.util.*;

public class triplets {

    // returns all triplets whose sum is equal to sum value
    public static List<List<Integer> > findTriplets(int[] nums, int sum)
    {
```

```
    /* Sort the elements */
    Arrays.sort(nums);

    List<List<Integer> > pair = new ArrayList<>();
    TreeSet<String> set = new TreeSet<String>();
    List<Integer> triplets = new ArrayList<>();

    /* Iterate over the array from the start and
       consider it as the first element*/
    for (int i = 0; i < nums.length - 2; i++) {

        // index of the first element in the
        // remaining elements
        int j = i + 1;

        // index of the last element
        int k = nums.length - 1;

        while (j < k) {

            if (nums[i] + nums[j] + nums[k] == sum) {

                String str = nums[i] + ":" + nums[j] + ":" + nums[k];

                if (!set.contains(str)) {

                    // To check for the unique triplet
                    triplets.add(nums[i]);
                    triplets.add(nums[j]);
                    triplets.add(nums[k]);
                    pair.add(triplets);
                    triplets = new ArrayList<>();
                    set.add(str);
                }

                j++; // increment the second value index
                k--; // decrement the third value index

            } else if (nums[i] + nums[j] + nums[k] < sum)
                j++;

            else // nums[i] + nums[j] + nums[k] > sum
                k--;
        }
    }
    return pair;
}
```

```
    public static void main(String[] args)
    {
        int[] nums = { 12, 3, 6, 1, 6, 9 };
        int sum = 24;

        List<List<Integer> > triplets = findTriplets(nums, sum);

        if (!triplets.isEmpty()) {
            System.out.println(triplets);
        } else {
            System.out.println("No triplets can be formed");
        }
    }
}
```

Output:

```
[[3, 9, 12], [6, 6, 12]]
```

Time Complexity: $O(n^2)$

Improved By : nik1996

Source

https://www.geeksforgeeks.org/unique-triplets-sum-given-value/

Chapter 3

Allocate minimum number of pages

Allocate minimum number of pages - GeeksforGeeks

Given number of pages in n different books and m students. The books are arranged in ascending order of number of pages. Every student is assigned to read some consecutive books. The task is to assign books in such a way that the maximum number of pages assigned to a student is minimum.

Example :

```
Input : pages[] = {12, 34, 67, 90}
        m = 2
Output : 113
Explanation:
There are 2 number of students. Books can be distributed
in following fashion :
  1) [12] and [34, 67, 90]
      Max number of pages is allocated to student
      2 with 34 + 67 + 90 = 191 pages
  2) [12, 34] and [67, 90]
      Max number of pages is allocated to student
      2 with 67 + 90 = 157 pages
  3) [12, 34, 67] and [90]
      Max number of pages is allocated to student
      1 with 12 + 34 + 67 = 113 pages

Of the 3 cases, Option 3 has the minimum pages = 113.
```

The idea is to use Binary Search. We fix a value for the number of pages as mid of current minimum and maximum. We initialize minimum and maximum as 0 and sum-of-all-pages

respectively. If a current mid can be a solution, then we search on the lower half, else we search in higher half.

Now the question arises, how to check if a mid value is feasible or not? Basically, we need to check if we can assign pages to all students in a way that the maximum number doesn't exceed current value. To do this, we sequentially assign pages to every student while the current number of assigned pages doesn't exceed the value. In this process, if the number of students becomes more than m, then the solution is not feasible. Else feasible.

Below is an implementation of above idea.

C++

```
 // C++ program for optimal allocation of pages
#include<bits/stdc++.h>
using namespace std;

// Utility function to check if current minimum value
// is feasible or not.
bool isPossible(int arr[], int n, int m, int curr_min)
{
    int studentsRequired = 1;
    int curr_sum = 0;

    // iterate over all books
    for (int i = 0; i < n; i++)
    {
        // check if current number of pages are greater
        // than curr_min that means we will get the result
        // after mid no. of pages
        if (arr[i] > curr_min)
            return false;

        // count how many students are required
        // to distribute curr_min pages
        if (curr_sum + arr[i] > curr_min)
        {
            // increment student count
            studentsRequired++;

            // update curr_sum
            curr_sum = arr[i];

            // if students required becomes greater
            // than given no. of students,return false
            if (studentsRequired > m)
                return false;
        }
```

```
        // else update curr_sum
        else
            curr_sum += arr[i];
    }
    return true;
}

// function to find minimum pages
int findPages(int arr[], int n, int m)
{
    long long sum = 0;

    // return -1 if no. of books is less than
    // no. of students
    if (n < m)
        return -1;

    // Count total number of pages
    for (int i = 0; i < n; i++)
        sum += arr[i];

    // initialize start as 0 pages and end as
    // total pages
    int start = 0, end = sum;
    int result = INT_MAX;

    // traverse until start <= end
    while (start <= end)
    {
        // check if it is possible to distribute
        // books by using mid as current minimum
        int mid = (start + end) / 2;
        if (isPossible(arr, n, m, mid))
        {
            // if yes then find the minimum distribution
            result = min(result, mid);

            // as we are finding minimum and books
            // are sorted so reduce end = mid -1
            // that means
            end = mid - 1;
        }

        else
            // if not possible means pages should be
            // increased so update start = mid + 1
            start = mid + 1;
    }
```

```
    // at-last return minimum no. of  pages
    return result;
}

// Drivers code
int main()
{
    //Number of pages in books
    int arr[] = {12, 34, 67, 90};
    int n = sizeof arr / sizeof arr[0];
    int m = 2; //No. of students

    cout << "Minimum number of pages = "
         << findPages(arr, n, m) << endl;
    return 0;
}
```

Java

```
 // Java program for optimal allocation of pages

public class GFG
{
    // Utility method to check if current minimum value
    // is feasible or not.
    static boolean isPossible(int arr[], int n, int m, int curr_min)
    {
        int studentsRequired = 1;
        int curr_sum = 0;

        // iterate over all books
        for (int i = 0; i < n; i++)
        {
            // check if current number of pages are greater
            // than curr_min that means we will get the result
            // after mid no. of pages
            if (arr[i] > curr_min)
                return false;

            // count how many students are required
            // to distribute curr_min pages
            if (curr_sum + arr[i] > curr_min)
            {
                // increment student count
                studentsRequired++;

                // update curr_sum
```

```
            curr_sum = arr[i];

            // if students required becomes greater
            // than given no. of students,return false
            if (studentsRequired > m)
                return false;
        }

        // else update curr_sum
        else
            curr_sum += arr[i];
    }
    return true;
}

// method to find minimum pages
static int findPages(int arr[], int n, int m)
{
    long sum = 0;

    // return -1 if no. of books is less than
    // no. of students
    if (n < m)
        return -1;

    // Count total number of pages
    for (int i = 0; i < n; i++)
        sum += arr[i];

    // initialize start as 0 pages and end as
    // total pages
    int start = 0, end = (int) sum;
    int result = Integer.MAX_VALUE;

    // traverse until start <= end
    while (start <= end)
    {
        // check if it is possible to distribute
        // books by using mid is current minimum
        int mid = (start + end) / 2;
        if (isPossible(arr, n, m, mid))
        {
            // if yes then find the minimum distribution
            result = Math.min(result, mid);

            // as we are finding minimum and books
            // are sorted so reduce end = mid -1
            // that means
```

```
                end = mid - 1;
            }

            else
                // if not possible means pages should be
                // increased so update start = mid + 1
                start = mid + 1;
        }

        // at-last return minimum no. of  pages
        return result;
    }

    // Driver Method
    public static void main(String[] args)
    {
        //Number of pages in books
        int arr[] = {12, 34, 67, 90};

        int m = 2; //No. of students

        System.out.println("Minimum number of pages = " +
                           findPages(arr, arr.length, m));
    }
}
```

C#

```
 // C# program for optimal
// allocation of pages
using System;

class GFG
{

// Utility function to check
// if current minimum value
// is feasible or not.
static bool isPossible(int []arr, int n,
                       int m, int curr_min)
{
    int studentsRequired = 1;
    int curr_sum = 0;

    // iterate over all books
    for (int i = 0; i < n; i++)
    {
        // check if current number of
```

```
        // pages are greater than curr_min
        // that means we will get the
        // result after mid no. of pages
        if (arr[i] > curr_min)
            return false;

        // count how many students
        // are required to distribute
        // curr_min pages
        if (curr_sum + arr[i] > curr_min)
        {
            // increment student count
            studentsRequired++;

            // update curr_sum
            curr_sum = arr[i];

            // if students required becomes
            // greater than given no. of
            // students, return false
            if (studentsRequired > m)
                return false;
        }

        // else update curr_sum
        else
            curr_sum += arr[i];
    }
    return true;
}

// function to find minimum pages
static int findPages(int []arr,
                     int n, int m)
{
    long sum = 0;

    // return -1 if no. of books
    // is less than no. of students
    if (n < m)
        return -1;

    // Count total number of pages
    for (int i = 0; i < n; i++)
        sum += arr[i];

    // initialize start as 0 pages
    // and end as total pages
```

```
    int start = 0, end = (int)sum;
    int result = int.MaxValue;

    // traverse until start <= end
    while (start <= end)
    {
        // check if it is possible to
        // distribute books by using
        // mid as current minimum
        int mid = (start + end) / 2;
        if (isPossible(arr, n, m, mid))
        {
            // if yes then find the
            // minimum distribution
            result = Math.Min(result, mid);

            // as we are finding minimum and
            // books are sorted so reduce
            // end = mid -1 that means
            end = mid - 1;
        }

        else
            // if not possible means pages
            // should be increased so update
            // start = mid + 1
            start = mid + 1;
    }

    // at-last return
    // minimum no. of pages
    return result;
}

// Drivers code
static public void Main ()
{

//Number of pages in books
int []arr = {12, 34, 67, 90};
int n = arr.Length;
int m = 2; // No. of students

Console.WriteLine("Minimum number of pages = " +
                          findPages(arr, n, m));
}
}
```

```
// This code is contributed by anuj_67.
```

Output :

```
Minimum number of pages = 113
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/allocate-minimum-number-pages/

Chapter 4

Array range queries for searching an element

Array range queries for searching an element - GeeksforGeeks

Given an array of N elements and Q queries of the form L R X. For each query, you have to output if the element X exists in the array between the indices L and R(included).

Prerequisite : Mo's Algorithms

Examples :

```
Input : N = 5
        arr = [1, 1, 5, 4, 5]
        Q = 3
        1 3 2
        2 5 1
        3 5 5
Output : No
         Yes
         Yes
Explanation :
For the first query, 2 does not exist between the indices 1 and 3.
For the second query, 1 exists between the indices 2 and 5.
For the third query, 5 exists between the indices 3 and 5.
```

Naive Approach :
The naive method would be to traverse the elements from L to R for each query, linearly searching for X. In the worst case, there can be N elements from L to R, hence the worst case time complexity for each query would be O(N). Therefore, for all the Q queries, the time complexity would turn out to be O(Q*N).

Using Union-Find Method :
This method checks only one element among all the consecutive equal values. If X is not equal to these values, then the algorithm skips all the other the other equal elements and continues traversal with the next different element. This algorithm is evidently useful only when there are consecutive equal elements in large amounts.

Algorithm :

1. Merge all the consecutive equal elements in one group.
2. While processing a query, start from R. Let index = R.
3. Compare a[index] with X. If they are equal, then print "Yes" and break out of traversing the rest of the range. Else, skip all the consecutive elements belonging to the group of a[index]. Index becomes equal to one less than the index of the root of this group.
4. Continue the above step either till X is found or till index becomes less than L.
5. If index becomes less than L, print "No".

Below is the C++ implementation of the above idea.

```
 // Program to determine if the element
// exists for different range queries
#include <bits/stdc++.h>
using namespace std;

// Structure to represent a query range
struct Query
{
    int L, R, X;
};

const int maxn = 100;

int root[maxn];

// Find the root of the group containing
// the element at index x
int find(int x)
{
    return x == root[x] ? x : root[x] =
                find(root[x]);
}

// merge the two groups containing elements
// at indices x and y into one group
int uni(int x, int y)
{
    int p = find(x), q = find(y);
```

```
    if (p != q) {
        root[p] = root[q];
    }
}

void initialize(int a[], int n, Query q[], int m)
{
    // make n subsets with every
    // element as its root
    for (int i = 0; i < n; i++)
        root[i] = i;

    // consecutive elements equal in value are
    // merged into one single group
    for (int i = 1; i < n; i++)
        if (a[i] == a[i - 1])
            uni(i, i - 1);
}

// Driver code
int main()
{
    int a[] = { 1, 1, 5, 4, 5 };
    int n = sizeof(a) / sizeof(a[0]);
    Query q[] = { { 0, 2, 2 }, { 1, 4, 1 },
                  { 2, 4, 5 } };
    int m = sizeof(q) / sizeof(q[0]);
    initialize(a, n, q, m);

    for (int i = 0; i < m; i++)
    {
        int flag = 0;
        int l = q[i].L, r = q[i].R, x = q[i].X;
        int p = r;

        while (p >= l)
        {

            // check if the current element in
            // consideration is equal to x or not
            // if it is equal, then x exists in the range
            if (a[p] == x)
            {
                flag = 1;
                break;
            }
            p = find(p) - 1;
        }
```

```
        // Print if x exists or not
        if (flag != 0)
            cout << x << " exists between [" << l
                 << ", " << r << "] " << endl;
        else
            cout << x << " does not exist between ["
                << l << ", " << r  << "] " << endl;
    }
}
```

Output:

```
2 does not exist between [0, 2]
1 exists between [1, 4]
5 exists between [2, 4]
```

Efficient Approach(Using Mo's Algorithm) :
Mo's algorithm is one of the finest applications for square root decomposition.
It is based on the basic idea of using the answer to the previous query to compute the answer for the current query. This is made possible because the Mo's algorithm is constructed in such a way that if F([L, R]) is known, then F([L + 1, R]), F([L – 1, R]), F([L, R + 1]) and F([L, R – 1]) can be computed easily, each in O(F) time.

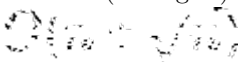
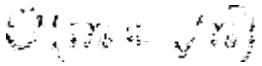
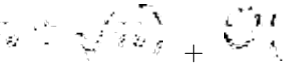
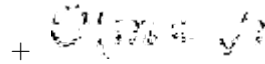
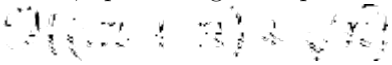
Answering queries in the order they are asked, then the time complexity is not improved to what is needed to be. To reduce the time complexity considerably, the queries are divided into blocks and then sorted. The exact algorithm to sort the queries is as follows :

- Denote BLOCK_SIZE = sqrt(N)
- All the queries with the same L/BLOCK_SIZE are put in the same block
- Within a block, the queries are sorted based on their R values
- The sort function thus compares two queries, Q1 and Q2 as follows:
  Q1 must come before Q2 if:
  1. L1/BLOCK_SIZE<L2/BLOCK_SIZE
  2. L1/BLOCK_SIZE=L2/BLOCK_SIZE and R1<R2

After sorting the queries, the next step is to compute the answer to the first query and consequently answer rest of the queries. To determine if a particular element exists or not, check the frequency of the element in that range. A non zero frequency confirms the existence of the element in that range.
To store the frequency of the elements, STL map has been used in the following code.
In the example given, first query after sorting the array of queries is {0, 2, 2}. Hash the frequencies of the elements in [0, 2] and then check the frequency of the element 2 from the map. Since, 2 occurs 0 times, print "No".
While processing the next query, which is {1, 4, 1} in this case, decrement the frequencies of the elements in the range [0, 1) and increment the frequencies of the elements in range

[3, 4]. This step gives the frequencies of elements in [1, 4] and it can easily be seen from the map that 1 exists in this range.

Time complexity :
The pre-processing part, that is sorting the queries takes O(m Log m) time.

The index variable for R changes at most  times throughout the run and that for L changes its value at most  times. Hence, processing all queries takes  +  =  time.

Below is the C++ implementation of the above idea :

```
 // CPP code to determine if the element
// exists for different range queries
#include <bits/stdc++.h>

using namespace std;

// Variable to represent block size.
// This is made global, so compare()
// of sort can use it.
int block;

// Structure to represent a query range
struct Query
{
    int L, R, X;
};

// Function used to sort all queries so
// that all queries of same block are
// arranged together and within a block,
// queries are sorted in increasing order
// of R values.
bool compare(Query x, Query y)
{
    // Different blocks, sort by block.
    if (x.L / block != y.L / block)
        return x.L / block < y.L / block;

    // Same block, sort by R value
    return x.R < y.R;
}

// Determines if the element is present for all
```

```
// query ranges. m is number of queries
// n is size of array a[].
void queryResults(int a[], int n, Query q[], int m)
{
    // Find block size
    block = (int)sqrt(n);

    // Sort all queries so that queries of same
    // blocks are arranged together.
    sort(q, q + m, compare);

    // Initialize current L, current R
    int currL = 0, currR = 0;

    // To store the frequencies of
    // elements of the given range
    map<int, int> mp;

    // Traverse through all queries
    for (int i = 0; i < m; i++) {

        // L and R values of current range
        int L = q[i].L, R = q[i].R, X = q[i].X;

        // Decrement frequencies of extra elements
        // of previous range. For example if previous
        // range is [0, 3] and current range is [2, 5],
        // then the frequencies of a[0] and a[1] are decremented
        while (currL < L)
        {
            mp[a[currL]]--;
            currL++;
        }

        // Increment frequencies of elements of current Range
        while (currL > L)
        {
            mp[a[currL - 1]]++;
            currL--;
        }
        while (currR <= R)
        {
            mp[a[currR]]++;
            currR++;
        }

        // Decrement frequencies of elements of previous
        // range.  For example when previous range is [0, 10]
```

```
        // and current range is [3, 8], then frequencies of
        // a[9] and a[10] are decremented
        while (currR > R + 1)
        {
            mp[a[currR - 1]]--;
            currR--;
        }

        // Print if X exists or not
        if (mp[X] != 0)
            cout << X << " exists between [" << L
                 << ", " << R << "] " << endl;
        else
            cout << X << " does not exist between ["
                 << L << ", " << R << "] " << endl;
    }
}

// Driver program
int main()
{
    int a[] = { 1, 1, 5, 4, 5 };
    int n = sizeof(a) / sizeof(a[0]);
    Query q[] = { { 0, 2, 2 }, { 1, 4, 1 }, { 2, 4, 5 } };
    int m = sizeof(q) / sizeof(q[0]);
    queryResults(a, n, q, m);
    return 0;
}
```

Output:

```
2 does not exist between [0, 2]
1 exists between [1, 4]
5 exists between [2, 4]
```

Source

https://www.geeksforgeeks.org/array-range-queries-for-searching-an-element/

Chapter 5

Best First Search (Informed Search)

Best First Search (Informed Search) - GeeksforGeeks

Prerequisites : BFS, DFS

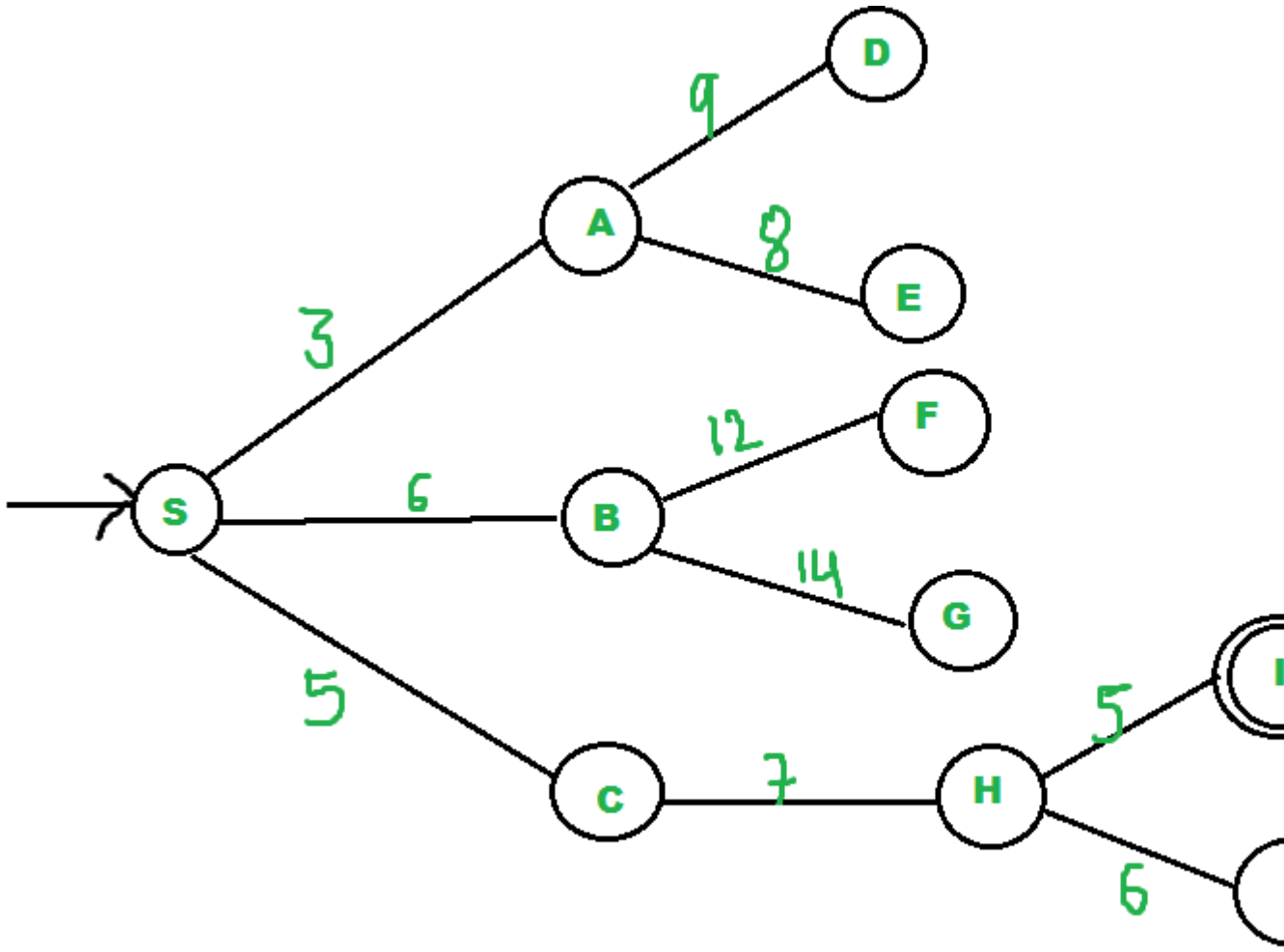
In BFS and DFS, when we are at a node, we can consider any of the adjacent as next node. So both BFS and DFS blindly explore paths without considering any cost function. The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.

We use a priority queue to store costs of nodes. So the implementation is a variation of BFS, we just need to change Queue to PriorityQueue.

```
// This pseudocode is adapted from below
// source:
// https://courses.cs.washington.edu/
Best-First-Search(Grah g, Node start)
    1) Create an empty PriorityQueue
       PriorityQueue pq;
    2) Insert "start" in pq.
       pq.insert(start)
    3) Until PriorityQueue is empty
          u = PriorityQueue.DeleteMin
          If u is the goal
             Exit
          Else
             Foreach neighbor v of u
                If v "Unvisited"
                    Mark v "Visited"
```

```
                    pq.insert(v)
                Mark v "Examined"
End procedure
```

Let us consider below example.




```
We start from source "S" and search for
goal "I" using given costs and Best
First search.

pq initially contains S
We remove s from and process unvisited
neighbors of S to pq.
pq now contains {A, C, B} (C is put
```

```
before B because C has lesser cost)

We remove A from pq and process unvisited
neighbors of A to pq.
pq now contains {C, B, E, D}

We remove C from pq and process unvisited
neighbors of C to pq.
pq now contains {B, H, E, D}

We remove B from pq and process unvisited
neighbors of B to pq.
pq now contains {H, E, D, F, G}

We remove H from pq.  Since our goal
"I" is a neighbor of H, we return.
```

Analysis :

- The worst case time complexity for Best First Search is O(n * Log n) where n is number of nodes. In worst case, we may have to visit all nodes before we reach goal. Note that priority queue is implemented using Min(or Max) Heap, and insert and remove operations take O(log n) time.
- Performance of the algorithm depends on how well the cost or evaluation function is designed.

Related Article:
A* Search Algorithm

Source

https://www.geeksforgeeks.org/best-first-search-informed-search/

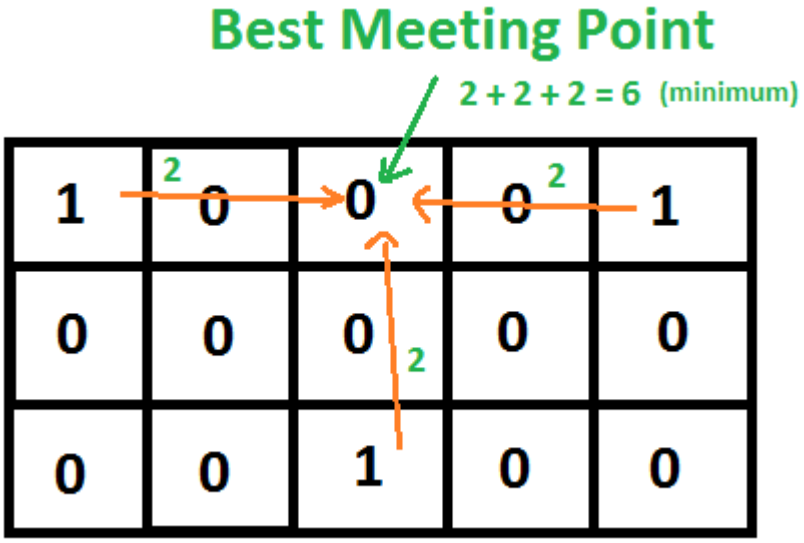
Chapter 6

Best meeting point in 2D binary array

Best meeting point in 2D binary array - GeeksforGeeks

You are given a 2D grid of values 0 or 1, where each 1 marks the home of someone in a group. And the group of two or more people wants to meet and minimize the total travel distance. They can meet anywhere means that there might be a home or not.

- The distance is calculated using Manhattan Distance, where distance(p1, p2) = |p2.x – p1.x| + |p2.y – p1.y|.
- Find the total distance that needs to be traveled to reach the best meeting point (Total distance traveled is minimum).



Examples:

```
Input : grid[][] = {{1, 0, 0, 0, 1},
                    {0, 0, 0, 0, 0},
                    {0, 0, 1, 0, 0}};
Output : 6
Best meeting point is (0, 2).
Total distance traveled is 2 + 2 + 2 = 6

Input : grid[3][5] = {{1, 0, 1, 0, 1},
                      {0, 1, 0, 0, 0},
                      {0, 1, 1, 0, 0}};
Output : 11
```

Steps :-
1) Store all horizontal and vertical positions of all group member.
2) Now sort it to find minimum middle position, which will be the best meeting point.
3) Find the distance of all members from best meeting point.

For example in above diagram, horizontal positions are {0, 2, 0} and vertical positions are {0, 2, 4}. After sorting both, we get {0, 0, 2} and {0, 2, 4}. Middle point is (0, 2).

Note : Even no. of 1's have two middle points, then also it works. Two middle points means it have two best meeting points always. Both cases will give same distance. So we will consider only one best meeting point to avoid the more overhead, Because our aim is to find the distance only.

```
 /* C++ program to find best meeting point in 2D array*/
#include <bits/stdc++.h>
using namespace std;
#define ROW 3
#define COL 5

int minTotalDistance(int grid[][COL]) {
    if (ROW == 0 || COL == 0)
         return 0;

    vector<int> vertical;
    vector<int> horizontal;

    // Find all members home's position
    for (int i = 0; i < ROW; i++) {
        for (int j = 0; j < COL; j++) {
            if (grid[i][j] == 1) {
                vertical.push_back(i);
                horizontal.push_back(j);
            }
```

```
        }
    }

    // Sort positions so we can find most
    // beneficial point
    sort(vertical.begin(),vertical.end());
    sort(horizontal.begin(),horizontal.end());

    // middle position will always beneficial
    // for all group members but it will be
    // sorted which we have alredy done
    int size = vertical.size()/2;
    int x = vertical[size];
    int y = horizontal[size];

    // Now find total distance from best meeting
    // point (x,y) using Manhattan Distance formula
    int distance = 0;
    for (int i = 0; i < ROW; i++)
        for (int j = 0; j < COL; j++)
            if (grid[i][j] == 1)
                distance += abs(x - i) + abs(y - j);

    return distance;
}

// Driver program to test above functions
int main() {
    int grid[ROW][COL] = {{1, 0, 1, 0, 1}, {0, 1, 0, 0, 0},{0, 1, 1, 0, 0}};
    cout << minTotalDistance(grid);
    return 0;
}
```

Output:

```
11
```

Source

https://www.geeksforgeeks.org/best-meeting-point-2d-binary-array/

Chapter 7

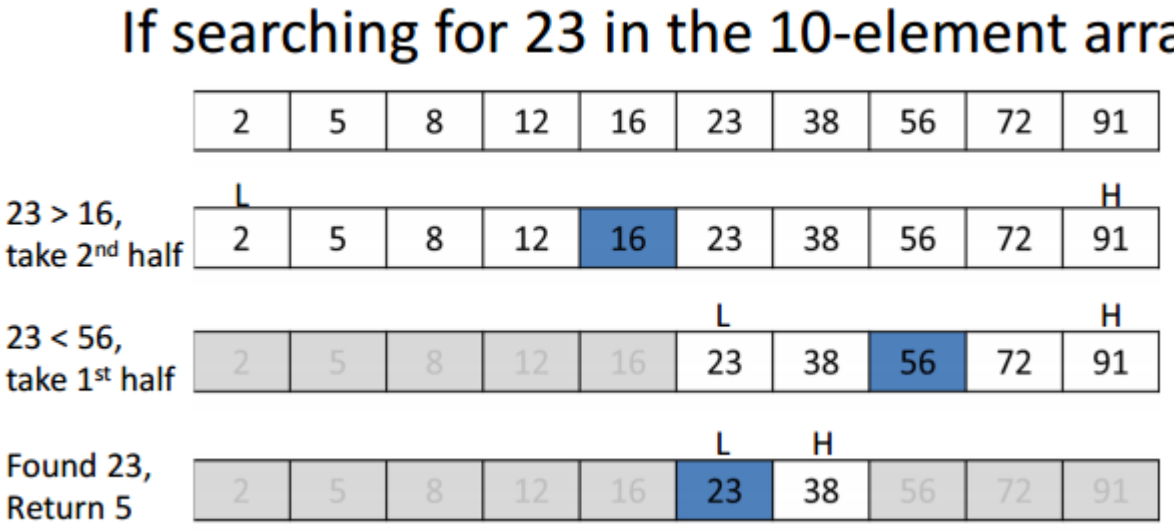
Binary Search

Binary Search - GeeksforGeeks

Given a sorted array arr[] of n elements, write a function to search a given element x in arr[].

A simple approach is to do **linear search.**The time complexity of above algorithm is O(n). Another approach to perform the same task is using Binary Search.

Binary Search: Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.



Example :
Image Source : http://www.nyckidd.com/bob/Linear%20Search%20and%20Binary%20Search_WorkingCopy.pdf

The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.

Recursive implementation of Binary Search

C/C++

```
 // C program to implement recursive Binary Search
#include <stdio.h>

// A recursive binary search function. It returns
// location of x in given array arr[l..r] is present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
   if (r >= l)
   {
        int mid = l + (r - l)/2;

        // If the element is present at the middle
        // itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid-1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid+1, r, x);
   }

   // We reach here when element is not
   // present in array
   return -1;
}

int main(void)
```

```
{
   int arr[] = {2, 3, 4, 10, 40};
   int n = sizeof(arr)/ sizeof(arr[0]);
   int x = 10;
   int result = binarySearch(arr, 0, n-1, x);
   (result == -1)? printf("Element is not present in array")
                 : printf("Element is present at index %d",
                                                 result);
   return 0;
}
```

Java

```
 // Java implementation of recursive Binary Search
class BinarySearch
{
    // Returns index of x if it is present in arr[l..
    // r], else return -1
    int binarySearch(int arr[], int l, int r, int x)
    {
        if (r>=l)
        {
            int mid = l + (r - l)/2;

            // If the element is present at the
            // middle itself
            if (arr[mid] == x)
               return mid;

            // If element is smaller than mid, then
            // it can only be present in left subarray
            if (arr[mid] > x)
               return binarySearch(arr, l, mid-1, x);

            // Else the element can only be present
            // in right subarray
            return binarySearch(arr, mid+1, r, x);
        }

        // We reach here when element is not present
        //  in array
        return -1;
    }

    // Driver method to test above
    public static void main(String args[])
    {
        BinarySearch ob = new BinarySearch();
```

```
        int arr[] = {2,3,4,10,40};
        int n = arr.length;
        int x = 10;
        int result = ob.binarySearch(arr,0,n-1,x);
        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at index " +
                                                   result);
    }
}
/* This code is contributed by Rajat Mishra */
```

Python

```
 # Python Program for recursive binary search.

# Returns index of x in arr if present, else -1
def binarySearch (arr, l, r, x):

    # Check base case
    if r >= l:

        mid = l + (r - l)/2

        # If element is present at the middle itself
        if arr[mid] == x:
            return mid

        # If element is smaller than mid, then it
        # can only be present in left subarray
        elif arr[mid] > x:
            return binarySearch(arr, l, mid-1, x)

        # Else the element can only be present
        # in right subarray
        else:
            return binarySearch(arr, mid+1, r, x)

    else:
        # Element is not present in the array
        return -1

# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
```

```
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print "Element is present at index %d" % result
else:
    print "Element is not present in array"
```

C#

```
 // C# implementation of recursive Binary Search
using System;

class GFG
{
    // Returns index of x if it is present in
    // arr[l..r], else return -1
    static int binarySearch(int []arr, int l,
                                  int r, int x)
    {
        if (r >= l)
        {
            int mid = l + (r - l)/2;

            // If the element is present at the
            // middle itself
            if (arr[mid] == x)
                return mid;

            // If element is smaller than mid, then
            // it can only be present in left subarray
            if (arr[mid] > x)
                return binarySearch(arr, l, mid-1, x);

            // Else the element can only be present
            // in right subarray
            return binarySearch(arr, mid+1, r, x);
        }

        // We reach here when element is not present
        // in array
        return -1;
    }

    // Driver method to test above
    public static void Main()
    {

        int []arr = {2, 3, 4, 10, 40};
```

```
        int n = arr.Length;
        int x = 10;

        int result = binarySearch(arr, 0, n-1, x);

        if (result == -1)
            Console.WriteLine("Element not present");
        else
            Console.WriteLine("Element found at index "
                                            + result);
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to implement
// recursive Binary Search

// A recursive binary search
// function. It returns location
// of x in given array arr[l..r]
// is present, otherwise -1
function binarySearch($arr, $l, $r, $x)
{
if ($r >= $l)
{
        $mid = $l + ($r - $l) / 2;

        // If the element is present
        // at the middle itself
        if ($arr[$mid] == $x)
            return floor($mid);

        // If element is smaller than
        // mid, then it can only be
        // present in left subarray
        if ($arr[$mid] > $x)
            return binarySearch($arr, $l,
                                $mid - 1, $x);

        // Else the element can only
        // be present in right subarray
        return binarySearch($arr, $mid + 1,
                            $r, $x);
}
```

```
// We reach here when element
// is not present in array
return -1;
}

// Driver Code
$arr = array(2, 3, 4, 10, 40);
$n = count($arr);
$x = 10;
$result = binarySearch($arr, 0, $n - 1, $x);
if(($result == -1))
echo "Element is not present in array";
else
echo "Element is present at index ",
                            $result;

// This code is contributed by anuj_67.
?>
```

Output :

```
Element is present at index 3
```

Iterative implementation of Binary Search

C/C++

```
 // C program to implement iterative Binary Search
#include <stdio.h>

// A iterative binary search function. It returns
// location of x in given array arr[l..r] if present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r)
    {
        int m = l + (r-l)/2;

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;
```

```
        // If x is smaller, ignore right half
        else
            r = m - 1;
    }

    // if we reach here, then element was
    // not present
    return -1;
}

int main(void)
{
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr)/ sizeof(arr[0]);
    int x = 10;
    int result = binarySearch(arr, 0, n-1, x);
    (result == -1)? printf("Element is not present"
                                    " in array")
               : printf("Element is present at "
                               "index %d", result);
    return 0;
}
```

Java

```
 // Java implementation of iterative Binary Search
class BinarySearch
{
    // Returns index of x if it is present in arr[],
    // else return -1
    int binarySearch(int arr[], int x)
    {
        int l = 0, r = arr.length - 1;
        while (l <= r)
        {
            int m = l + (r-l)/2;

            // Check if x is present at mid
            if (arr[m] == x)
                return m;

            // If x greater, ignore left half
            if (arr[m] < x)
                l = m + 1;

            // If x is smaller, ignore right half
            else
                r = m - 1;
```

```
        }

        // if we reach here, then element was
        // not present
        return -1;
    }

    // Driver method to test above
    public static void main(String args[])
    {
        BinarySearch ob = new BinarySearch();
        int arr[] = {2, 3, 4, 10, 40};
        int n = arr.length;
        int x = 10;
        int result = ob.binarySearch(arr, x);
        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at " +
                               "index " + result);
    }
}
```

Python

```
 # Python code to implement iterative Binary
# Search.

# It returns location of x in given array arr
# if present, else returns -1
def binarySearch(arr, l, r, x):

    while l <= r:

        mid = l + (r - l)/2;

        # Check if x is present at mid
        if arr[mid] == x:
            return mid

        # If x is greater, ignore left half
        elif arr[mid] < x:
            l = mid + 1

        # If x is smaller, ignore right half
        else:
            r = mid - 1
```

```
    # If we reach here, then the element
    # was not present
    return -1


# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print "Element is present at index %d" % result
else:
    print "Element is not present in array"
```

C#

```
 // C# implementation of iterative Binary Search
using System;

class GFG
{
    // Returns index of x if it is present in arr[],
    // else return -1
    static int binarySearch(int []arr, int x)
    {
        int l = 0, r = arr.Length - 1;
        while (l <= r)
        {
            int m = l + (r-l)/2;

            // Check if x is present at mid
            if (arr[m] == x)
                return m;

            // If x greater, ignore left half
            if (arr[m] < x)
                l = m + 1;

            // If x is smaller, ignore right half
            else
                r = m - 1;
        }

        // if we reach here, then element was
        // not present
```

```
        return -1;
    }

    // Driver method to test above
    public static void Main()
    {
        int []arr = {2, 3, 4, 10, 40};
        int n = arr.Length;
        int x = 10;
        int result = binarySearch(arr, x);
        if (result == -1)
            Console.WriteLine("Element not present");
        else
            Console.WriteLine("Element found at " +
                              "index " + result);
    }
}
// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to implement
// iterative Binary Search

// A iterative binary search
// function. It returns location
// of x in given array arr[l..r]
// if present, otherwise -1
function binarySearch($arr, $l,
                      $r, $x)
{
    while ($l <= $r)
    {
        $m = $l + ($r - $l) / 2;

        // Check if x is present at mid
        if ($arr[$m] == $x)
            return floor($m);

        // If x greater, ignore
        // left half
        if ($arr[$m] < $x)
            $l = $m + 1;

        // If x is smaller,
        // ignore right half
        else
```

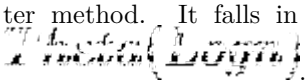
```
            $r = $m - 1;
    }

    // if we reach here, then
    // element was not present
    return -1;
}

// Driver Code
$arr = array(2, 3, 4, 10, 40);
$n = count($arr);
$x = 10;
$result = binarySearch($arr, 0,
                       $n - 1, $x);
if(($result == -1))
echo "Element is not present in array";
else
echo "Element is present at index ",
                            $result;

// This code is contributed by anuj_67.
?>
```

Output :

```
Element is present at index 3
```

Time Complexity:
The time complexity of Binary Search can be written as

```
T(n) = T(n/2) + c
```

The above recurrence can be solved either using Recurrence T ree method or Master method. It falls in case II of Master Method and solution of the recurrence is  .

Auxiliary Space: O(1) in case of iterative implementation. In case of recursive implementation, O(Logn) recursion call stack space.

Algorithmic Paradigm: Decrease and Conquer.

Interesting articles based on Binary Search.

- The Ubiquitous Binary Search
- Interpolation search vs Binary search
- Find the minimum element in a sorted and rotated array

- Find a peak element
- Find a Fixed Point in a given array
- Count the number of occurrences in a sorted array
- Median of two sorted arrays
- Floor and Ceiling in a sorted array
- Find the maximum element in an array which is first increasing and then decreasing

Coding Practice Questions on Binary Search
Recent Articles on Binary Search.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/binary-search/

Chapter 8

Binary Search (bisect) in Python

Binary Search (bisect) in Python - GeeksforGeeks

Binary Search is a technique used to search element in a sorted list. In this article, we will looking at library functions to do Binary Search.

Finding first occurrence of an element.

> bisect.bisect_left(a, x, lo=0, hi=len(a)) : Returns leftmost insertion point of x in a sorted list. Last two parameters are optional, they are used to search in sublist.

```
 # Python code to demonstrate working
# of binary search in library
from bisect import bisect_left

def BinarySearch(a, x):
    i = bisect_left(a, x)
    if i != len(a) and a[i] == x:
        return i
    else:
        return -1

a  = [1, 2, 4, 4, 8]
x = int(4)
res = BinarySearch(a, x)
if res == -1:
    print(x, "is absent")
else:
    print("First occurrence of", x, "is present at", res)
```

Output:

```
First occurrence of 4 is present at 2
```

Finding greatest value smaller than x.

```
 # Python code to demonstrate working
# of binary search in library
from bisect import bisect_left

def BinarySearch(a, x):
    i = bisect_left(a, x)
    if i:
        return (i-1)
    else:
        return -1

# Driver code
a  = [1, 2, 4, 4, 8]
x = int(7)
res = BinarySearch(a, x)
if res == -1:
    print("No value smaller than ", x)
else:
    print("Largest value smaller than ", x, " is at index ", res)
```

Output:

```
Largest value smaller than  7  is at index  3
```

Finding rightmost occurrence

bisect.bisect_right(a, x, lo=0, hi=len(a)) Returns rightmost insertion point of x in a sorted list a. Last two parameters are optional, they are used to search in sublist.

```
 # Python code to demonstrate working
# of binary search in library
from bisect import bisect_right

def BinarySearch(a, x):
    i = bisect_right(a, x)
    if i != len(a)+1 and a[i-1] == x:
        return (i-1)
```

```
    else:
        return -1

a  = [1, 2, 4, 4]
x = int(4)
res = BinarySearch(a, x)
if res == -1:
    print(x, "is absent")
else:
    print("Last occurrence of", x, "is present at", res)
```

Output:

```
Last occurrence of 4 is present at 3
```

Please refer Binary Search for writing your own Binary Search code.

Reference :
https://docs.python.org/3/library/bisect.html

Source

https://www.geeksforgeeks.org/binary-search-bisect-in-python/

Chapter 9

Binary Search for Rational Numbers without using floating point arithmetic

Binary Search for Rational Numbers without using floating point arithmetic - GeeksforGeeks

A rational is represented as p/qb, for example 2/3. Given a sorted array of rational numbers, how to search an element using Binary Search. Use of floating point arithmetic is not allowed.

Example:

```
Input:  arr[] = {1/5, 2/3, 3/2, 13/2}
        x = 3/2
Output: Found at index 2
```

We strongly recommend you to minimize your browser and try this yourself first.

To compare two rational numbers p/q and r/s, we can compare p*s with q*r.

```
 // C program for Binary Search for Rationalnal Numbers
// without using floating point arithmetic
#include <stdio.h>

struct Rational
{
    int p;
    int q;
};

// Utility function to compare two Rationalnal numbers
```

```
// 'a' and 'b'. It returns
// 0 --> When 'a' and 'b' are same
// 1 --> When 'a' is greater
//-1 --> When 'b' is greate
int compare(struct Rational a, struct Rational b)
{
    // If a/b == c/d  then  a*d = b*c:
    // method to ignore division
    if (a.p * b.q == a.q * b.p)
        return 0;
    if (a.p * b.q > a.q * b.p)
        return 1;
    return -1;
}

// Returns index of x in arr[l..r] if it is present, else
// returns -1. It mainly uses Binary Search.
int binarySearch(struct Rational arr[], int l, int r,
                 struct Rational x)
{
   if (r >= l)
   {
        int mid = l + (r - l)/2;

        // If the element is present at the middle itself
        if (compare(arr[mid], x) == 0)  return mid;

        // If element is smaller than mid, then it can
        // only be present in left subarray
        if (compare(arr[mid], x) > 0)
            return binarySearch(arr, l, mid-1, x);

        // Else the element can only be present in right
        // subarray
        return binarySearch(arr, mid+1, r, x);
   }

   return -1;
}

// Driver method
int main()
{
    struct Rational arr[] = {{1, 5}, {2, 3}, {3, 2}, {13, 2}};
    struct Rational x = {3, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Element found at index %d",
            binarySearch(arr, 0, n-1, x));
```

```
}
```

Output:

```
Element found at index 2
```

Thanks to Utkarsh Trivedi for suggesting above solution.

Improved By : tumse_na_ho_payega

Source

https://www.geeksforgeeks.org/binary-search-for-rational-numbers-without-using-floating-point-arithmetic/

Chapter 10

Binary Search on Singly Linked List

Binary Search on Singly Linked List - GeeksforGeeks

Given a singly linked list and a key, find key using binary search approach.
To perform a Binary search based on Divide and Conquer Algorithm, determination of the middle element is important. Binary Search is usually fast and efficient for arrays because accessing the middle index between two given indices is easy and fast(Time Complexity O(1)). But memory allocation for the singly linked list is dynamic and non-contiguous, which makes finding the middle element difficult. One approach could be of using skip list, one could be traversing the linked list using one pointer.

Prerequisite : Finding middle of a linked list.

Note: The approach and implementation provided below are to show how Binary Search can be implemented on a linked list. The implementation takes O(n) time.

Approach :

- Here, start node(set to Head of list), and the last node(set to NULL initially) are given.
- Middle is calculated using two pointers approach.
- If middle's data matches the required value of search, return it.
- Else if midele's data < value, move to upper half(setting last to midele's next).
- Else go to lower half(setting last to middle).
- The condition to come out is, either element found or entire list is traversed. When entire list is traversed, last points to start i.e. last -> next == start.

In main function, function **InsertAtHead** inserts value at the beginning of linked list. Inserting such values(for sake of simplicity) so that the list created is sorted.

Examples :

```
Input : Enter value to search : 7
Output : Found

Input : Enter value to search : 12
Output : Not Found
```

```
 // CPP code to implement binary search
// on Singly Linked List
#include<stdio.h>
#include<stdlib.h>

struct Node
{
    int data;
    struct Node* next;
};

Node *newNode(int x)
{
    struct Node* temp = new Node;
    temp->data = x;
    temp->next = NULL;
    return temp;
}

// function to find out middle element
struct Node* middle(Node* start, Node* last)
{
    if (start == NULL)
        return NULL;

    struct Node* slow = start;
    struct Node* fast = start -> next;

    while (fast != last)
    {
        fast = fast -> next;
        if (fast != last)
        {
            slow = slow -> next;
            fast = fast -> next;
        }
    }

    return slow;
}
```

```
// Function for implementing the Binary
// Search on linked list
struct Node* binarySearch(Node *head, int value)
{
    struct Node* start = head;
    struct Node* last = NULL;

    do
    {
        // Find middle
        Node* mid = middle(start, last);

        // If middle is empty
        if (mid == NULL)
            return NULL;

        // If value is present at middle
        if (mid -> data == value)
            return mid;

        // If value is more than mid
        else if (mid -> data < value)
            start = mid -> next;

        // If the value is less than mid.
        else
            last = mid;

    } while (last == NULL ||
             last -> next != start);

    // value not present
    return NULL;
}

// Driver Code
int main()
{
    Node *head = newNode(1);
    head->next = newNode(4);
    head->next->next = newNode(7);
    head->next->next->next = newNode(8);
    head->next->next->next->next = newNode(9);
    head->next->next->next->next->next = newNode(10);
    int value = 7;
    if (binarySearch(head, value) == NULL)
        printf("Value not present\n");
    else
```

```
        printf("Present");
    return 0;
}
```

Output:

```
Present
```

Time Complexity : O(n)

Improved By : sunny94

Source

https://www.geeksforgeeks.org/binary-search-on-singly-linked-list/

Chapter 11

Binary Search using pthread

Binary Search using pthread - GeeksforGeeks

Binary search is a popular method of searching in a sorted array or list. It simply divides the list into two halves and discard the half which has zero probability of having the key. On dividing we check the mid point for the key and uses the lower half if key is less than mid point and upper half if key is greater than mid point. Binary search has time complexity of O(log(n)).

Binary search can also be implemented using **multi-threading** where we utilizes the cores of processor by providing each thread a portion of list to search for the key.

Number of threads depends upon the number of cores your processor has and its better to create one thread for each core.

Examples:

```
Input :  list = 1, 5, 7, 10, 12, 14, 15, 18, 20, 22, 25, 27, 30, 64, 110, 220
         key = 7
Output : 7 found in list

Input :  list = 1, 5, 7, 10, 12, 14, 15, 18, 20, 22, 25, 27, 30, 64, 110, 220
         key = 111
Output : 111 not found in list
```

Note – It is advised to execute the program in Linux based system.
Compile in linux using following code:

```
g++ -pthread program_name.cpp
```

```
 // CPP Program to perform binary search using pthreads
#include <iostream>
```

```
using namespace std;

// size of array
#define MAX 16

// maximum number of threads
#define MAX_THREAD 4

// array to be searched
int a[] = { 1, 5, 7, 10, 12, 14, 15, 18, 20, 22, 25, 27, 30, 64, 110, 220 };

// key that needs to be searched
int key = 110;
bool found = false;
int part = 0;

void* binary_search(void* arg)
{

    // Each thread checks 1/4 of the array for the key
    int thread_part = part++;
    int mid;

    int low = thread_part * (MAX / 4);
    int high = (thread_part + 1) * (MAX / 4);

    // search for the key until low < high
    // or key is found in any portion of array
    while (low < high && !found)  {

        // normal iterative binary search algorithm
        mid = (high - low) / 2 + low;

        if (a[mid] == key)  {
            found = true;
            break;
        }

        else if (a[mid] > key)
            high = mid - 1;
        else
            low = mid + 1;
    }
}


// Driver Code
int main()
```

```
{
    pthread_t threads[MAX_THREAD];

    for (int i = 0; i < MAX_THREAD; i++)
        pthread_create(&threads[i], NULL, binary_search, (void*)NULL);

    for (int i = 0; i < MAX_THREAD; i++)
        pthread_join(threads[i], NULL);

    // key found in array
    if (found)
        cout << key << " found in array" << endl;

    // key not found in array
    else
        cout << key << " not found in array" << endl;

    return 0;
}
```

Output:

```
110 found in array
```

Source

https://www.geeksforgeeks.org/binary-search-using-pthread/

Chapter 12

Binary search in sorted vector of pairs

Binary search in sorted vector of pairs - GeeksforGeeks

How to apply STL binary_search to vector of pairs(key, value), given that vector is sorted by its first value(key)

struct compare in the code contains two function which compares the key(searching element) with the first element in the vector

```
 /* C++ code to demonstrate how Binary Search
   can be applied on a vector of pairs */
#include <algorithm> // binary search
#include <iostream>
#include <vector>
using namespace std;

struct compare {
    bool operator()(const pair<int, int>& value,
                                const int& key)
    {
        return (value.first < key);
    }
    bool operator()(const int& key,
                    const pair<int, int>& value)
    {
        return (key < value.first);
    }
};

int main()
{
```

```
    // intializing the vector of pairs
    vector<pair<int, int> > vect;

    // insertion of pairs (key, value) in vector vect
    vect.push_back(make_pair(1, 20));
    vect.push_back(make_pair(3, 42));
    vect.push_back(make_pair(4, 36));
    vect.push_back(make_pair(2, 80));
    vect.push_back(make_pair(7, 50));
    vect.push_back(make_pair(9, 20));
    vect.push_back(make_pair(3, 29));

    // sorting the vector according to key
    sort(vect.begin(), vect.end());

    // printing the sorted vector
    cout << "KEY" << '\t' << "ELEMENT" << endl;
    for (pair<int, int>& x : vect)
        cout << x.first << '\t' << x.second << endl;

    // searching for the key element 3
    cout << "search for key 3 in vector" << endl;
    if (binary_search(vect.begin(), vect.end(),
                                  3, compare()))
        cout << "Element found";
    else
        cout << "Element not found";

    return 0;
}
```

Output:

```
KEY    ELEMENT
1    20
2    80
3    29
3    42
4    36
7    50
9    20
search for key 3 in vector
Element found
```

The above binary_search operation has time complexity **O(lg n)**

Source

https://www.geeksforgeeks.org/binary-search-sorted-vector-pairs/

Chapter 13

C Program for Binary Search (Recursive and Iterative)

C Program for Binary Search (Recursive and Iterative) - GeeksforGeeks

We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.

Recursive :

Source

https://www.geeksforgeeks.org/c-program-for-binary-search-recursive-and-iterative/

C/C++

```
 #include <stdio.h>

// A recursive binary search function. It returns location of x in
// given array arr[l..r] is present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
   if (r >= l)
   {
        int mid = l + (r - l)/2;
```

```
        // If the element is present at the middle itself
        if (arr[mid] == x)  return mid;

        // If element is smaller than mid, then it can only be present
        // in left subarray
        if (arr[mid] > x) return binarySearch(arr, l, mid-1, x);

        // Else the element can only be present in right subarray
        return binarySearch(arr, mid+1, r, x);
   }

   // We reach here when element is not present in array
   return -1;
}

int main(void)
{
   int arr[] = {2, 3, 4, 10, 40};
   int n = sizeof(arr)/ sizeof(arr[0]);
   int x = 10;
   int result = binarySearch(arr, 0, n-1, x);
   (result == -1)? printf("Element is not present in array")
                 : printf("Element is present at index %d", result);
   return 0;
}
```

Iterative

C/C++

```
 #include <stdio.h>

// A iterative binary search function. It returns location of x in
// given array arr[l..r] if present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
  while (l <= r)
  {
    int m = l + (r-l)/2;

    // Check if x is present at mid
    if (arr[m] == x)
        return m;

    // If x greater, ignore left half
    if (arr[m] < x)
        l = m + 1;
```

```
    // If x is smaller, ignore right half
    else
         r = m - 1;
  }

  // if we reach here, then element was not present
  return -1;
}

int main(void)
{
   int arr[] = {2, 3, 4, 10, 40};
   int n = sizeof(arr)/ sizeof(arr[0]);
   int x = 10;
   int result = binarySearch(arr, 0, n-1, x);
   (result == -1)? printf("Element is not present in array")
                 : printf("Element is present at index %d", result);
   return 0;
}
```

Please refer complete article on Binary Search for more details!

Chapter 14

Ceiling in a sorted array

Ceiling in a sorted array - GeeksforGeeks

Given a sorted array and a value x, the ceiling of x is the smallest element in array greater than or equal to x, and the floor is the greatest element smaller than or equal to x. Assume than the array is sorted in non-decreasing order. Write efficient functions to find floor and ceiling of x.

Examples :

```
For example, let the input array be {1, 2, 8, 10, 10, 12, 19}
For x = 0:    floor doesn't exist in array,  ceil  = 1
For x = 1:    floor  = 1,  ceil  = 1
For x = 5:    floor  = 2,  ceil  = 8
For x = 20:   floor  = 19,  ceil doesn't exist in array
```

In below methods, we have implemented only ceiling search functions. Floor search can be implemented in the same way.

Method 1 (Linear Search)
Algorithm to search ceiling of x:
1) If x is smaller than or equal to the first element in array then return 0(index of first element)
2) Else Linearly search for an index i such that x lies between arr[i] and arr[i+1].
3) If we do not find an index i in step 2, then return -1

C

```
 #include<stdio.h>

/* Function to get index of ceiling of x in arr[low..high] */
int ceilSearch(int arr[], int low, int high, int x)
{
```

```
  int i;

  /* If x is smaller than or equal to first element,
    then return the first element */
  if(x <= arr[low])
    return low;

  /* Otherwise, linearly search for ceil value */
  for(i = low; i < high; i++)
  {
    if(arr[i] == x)
      return i;

    /* if x lies between arr[i] and arr[i+1] including
       arr[i+1], then return arr[i+1] */
    if(arr[i] < x && arr[i+1] >= x)
       return i+1;
  }

  /* If we reach here then x is greater than the last element
    of the array,  return -1 in this case */
  return -1;
}


/* Driver program to check above functions */
int main()
{
   int arr[] = {1, 2, 8, 10, 10, 12, 19};
   int n = sizeof(arr)/sizeof(arr[0]);
   int x = 3;
   int index = ceilSearch(arr, 0, n-1, x);
   if(index == -1)
     printf("Ceiling of %d doesn't exist in array ", x);
   else
     printf("ceiling of %d is %d", x, arr[index]);
   getchar();
   return 0;
}
```

Java

```
 class Main
{
    /* Function to get index of ceiling
       of x in arr[low..high] */
    static int ceilSearch(int arr[], int low, int high, int x)
    {
```

```
      int i;

      /* If x is smaller than or equal to first
         element,then return the first element */
      if(x <= arr[low])
        return low;

      /* Otherwise, linearly search for ceil value */
      for(i = low; i < high; i++)
      {
        if(arr[i] == x)
          return i;

        /* if x lies between arr[i] and arr[i+1]
        including arr[i+1], then return arr[i+1] */
        if(arr[i] < x && arr[i+1] >= x)
           return i+1;
      }

      /* If we reach here then x is greater than the
      last element of the array,  return -1 in this case */
      return -1;
    }


    /* Driver program to check above functions */
    public static void main (String[] args)
    {
       int arr[] = {1, 2, 8, 10, 10, 12, 19};
       int n = arr.length;
       int x = 3;
       int index = ceilSearch(arr, 0, n-1, x);
       if(index == -1)
         System.out.println("Ceiling of "+x+" doesn't exist in array");
       else
         System.out.println("ceiling of "+x+" is "+arr[index]);
    }
}
```

Python3

```
 # Function to get index of ceiling of x in arr[low..high] */
def ceilSearch(arr, low, high, x):

    # If x is smaller than or equal to first element,
    # then return the first element */
    if x <= arr[low]:
        return low
```

```
    # Otherwise, linearly search for ceil value */
    i = low
    for i in range(high):
        if arr[i] == x:
            return i

        # if x lies between arr[i] and arr[i+1] including
        # arr[i+1], then return arr[i+1] */
        if arr[i] < x and arr[i+1] >= x:
            return i+1

    # If we reach here then x is greater than the last element
    # of the array,  return -1 in this case */
    return -1

# Driver program to check above functions */
arr = [1, 2, 8, 10, 10, 12, 19]
n = len(arr)
x = 3
index = ceilSearch(arr, 0, n-1, x);

if index == -1:
    print ("Ceiling of %d doesn't exist in array "% x)
else:
    print ("ceiling of %d is %d"%(x, arr[index]))

# This code is contributed by Shreyanshi Arun
```

C#

```
 // C# program to find celing
// in a sorted array
using System;

class GFG {

    // Function to get index of ceiling
    // of x in arr[low..high]
    static int ceilSearch(int[] arr, int low,
                          int high, int x)
    {
        int i;

        // If x is smaller than or equal
        // to first element, then return
        // the first element
        if (x <= arr[low])
```

```
            return low;

        // Otherwise, linearly search
        // for ceil value
        for (i = low; i < high; i++) {
            if (arr[i] == x)
                return i;

            /* if x lies between arr[i] and
            arr[i+1] including arr[i+1],
            then return arr[i+1] */
            if (arr[i] < x && arr[i + 1] >= x)
                return i + 1;
        }

        /* If we reach here then x is
        greater than the last element
        of the array, return -1 in
        this case */
        return -1;
    }

    // Driver code
    public static void Main()
    {
        int[] arr = { 1, 2, 8, 10, 10, 12, 19 };
        int n = arr.Length;
        int x = 3;
        int index = ceilSearch(arr, 0, n - 1, x);

        if (index == -1)
            Console.Write("Ceiling of " + x +
                     " doesn't exist in array");
        else
            Console.Write("ceiling of " + x +
                          " is " + arr[index]);
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// Function to get index of
// ceiling of x in arr[low..high]
function ceilSearch($arr, $low, $high, $x)
{
```

```
    // If x is smaller than or equal
    // to first element, then return
    // the first element
    if($x <= $arr[$low])
        return $low;

    // Otherwise, linearly search
    // for ceil value
    for($i = $low; $i < $high; $i++)
    {
        if($arr[$i] == $x)
            return $i;

        // if x lies between arr[i] and
        // arr[i+1] including arr[i+1],
        // then return arr[i+1]
        if($arr[$i] < $x &&
           $arr[$i + 1] >= $x)
            return $i + 1;
    }

    // If we reach here then x is greater
    // than the last element of the array,
    // return -1 in this case
    return -1;
}

// Driver Code
$arr = array(1, 2, 8, 10, 10, 12, 19);
$n = sizeof($arr);
$x = 3;
$index = ceilSearch($arr, 0, $n - 1, $x);
if($index == -1)
    echo("Ceiling of " . $x .
         " doesn't exist in array ");
else
    echo("ceiling of " . $x . " is " .
                         $arr[$index]);

// This code is contributed by Ajit.
?>
```

Output :

```
ceiling of 3 is 8
```

Time Complexity : O(n)

Method 2 (Binary Search)
Instead of using linear search, binary search is used here to find out the index. Binary search reduces time complexity to O(Logn).

C

```
 #include<stdio.h>

/* Function to get index of ceiling of x in arr[low..high]*/
int ceilSearch(int arr[], int low, int high, int x)
{
  int mid;

  /* If x is smaller than or equal to the first element,
    then return the first element */
  if(x <= arr[low])
    return low;

  /* If x is greater than the last element, then return -1 */
  if(x > arr[high])
    return -1;

  /* get the index of middle element of arr[low..high]*/
  mid = (low + high)/2;  /* low + (high - low)/2 */

  /* If x is same as middle element, then return mid */
  if(arr[mid] == x)
    return mid;

  /* If x is greater than arr[mid], then either arr[mid + 1]
    is ceiling of x or ceiling lies in arr[mid+1...high] */
  else if(arr[mid] < x)
  {
    if(mid + 1 <= high && x <= arr[mid+1])
      return mid + 1;
    else
      return ceilSearch(arr, mid+1, high, x);
  }

  /* If x is smaller than arr[mid], then either arr[mid]
     is ceiling of x or ceiling lies in arr[mid-1...high] */
  else
  {
    if(mid - 1 >= low && x > arr[mid-1])
      return mid;
    else
      return ceilSearch(arr, low, mid - 1, x);
```

```
  }
}

/* Driver program to check above functions */
int main()
{
   int arr[] = {1, 2, 8, 10, 10, 12, 19};
   int n = sizeof(arr)/sizeof(arr[0]);
   int x = 20;
   int index = ceilSearch(arr, 0, n-1, x);
   if(index == -1)
     printf("Ceiling of %d doesn't exist in array ", x);
   else
     printf("ceiling of %d is %d", x, arr[index]);
   getchar();
   return 0;
}
```

Java

```
 class Main
{
    /* Function to get index of
       ceiling of x in arr[low..high]*/
    static int ceilSearch(int arr[], int low, int high, int x)
    {
      int mid;

      /* If x is smaller than or equal to the
         first element, then return the first element */
      if(x <= arr[low])
        return low;

      /* If x is greater than the last
         element, then return -1 */
      if(x > arr[high])
        return -1;

      /* get the index of middle element
         of arr[low..high]*/
      mid = (low + high)/2;  /* low + (high - low)/2 */

      /* If x is same as middle element,
         then return mid */
      if(arr[mid] == x)
        return mid;

      /* If x is greater than arr[mid], then
```

```
         either arr[mid + 1] is ceiling of x or
         ceiling lies in arr[mid+1...high] */
      else if(arr[mid] < x)
      {
        if(mid + 1 <= high && x <= arr[mid+1])
          return mid + 1;
        else
          return ceilSearch(arr, mid+1, high, x);
      }

      /* If x is smaller than arr[mid],
         then either arr[mid] is ceiling of x
         or ceiling lies in arr[mid-1...high] */
      else
      {
        if(mid - 1 >= low && x > arr[mid-1])
          return mid;
        else
          return ceilSearch(arr, low, mid - 1, x);
      }
    }


    /* Driver program to check above functions */
    public static void main (String[] args)
    {
       int arr[] = {1, 2, 8, 10, 10, 12, 19};
       int n = arr.length;
       int x = 8;
       int index = ceilSearch(arr, 0, n-1, x);
       if(index == -1)
         System.out.println("Ceiling of "+x+" doesn't exist in array");
       else
         System.out.println("ceiling of "+x+" is "+arr[index]);
    }
}
```

Python3

```
 # Function to get index of ceiling of x in arr[low..high]*/
def ceilSearch(arr, low, high, x):

    # If x is smaller than or equal to the first element,
    # then return the first element */
    if x <= arr[low]:
        return low

    # If x is greater than the last element, then return -1 */
```

```
    if x > arr[high]:
        return -1

    # get the index of middle element of arr[low..high]*/
    mid = (low + high)/2;  # low + (high - low)/2 */

    # If x is same as middle element, then return mid */
    if arr[mid] == x:
        return mid

    # If x is greater than arr[mid], then either arr[mid + 1]
    # is ceiling of x or ceiling lies in arr[mid+1...high] */
    elif arr[mid] < x:
        if mid + 1 <= high and x <= arr[mid+1]:
            return mid + 1
        else:
            return ceilSearch(arr, mid+1, high, x)

    # If x is smaller than arr[mid], then either arr[mid]
    # is ceiling of x or ceiling lies in arr[mid-1...high] */
    else:
        if mid - 1 >= low and x > arr[mid-1]:
            return mid
        else:
            return ceilSearch(arr, low, mid - 1, x)

# Driver program to check above functions */
arr = [1, 2, 8, 10, 10, 12, 19]
n = len(arr)
x = 20
index = ceilSearch(arr, 0, n-1, x);

if index == -1:
    print ("Ceiling of %d doesn't exist in array "% x)
else:
    print ("ceiling of %d is %d"%(x, arr[index]))

# This code is contributed by Shreyanshi Arun
```

C#

```
 // C# program to find celing
// in a sorted array
using System;

class GFG {

    // Function to get index of ceiling
```

```
// of x in arr[low..high]
static int ceilSearch(int[] arr, int low,
                          int high, int x)
{
    int mid;

    // If x is smaller than or equal
    // to the first element, then
    // return the first element.
    if (x <= arr[low])
        return low;

    // If x is greater than the last
    // element, then return -1
    if (x > arr[high])
        return -1;

    // get the index of middle
    // element of arr[low..high]
    mid = (low + high) / 2;
    // low + (high - low)/2

    // If x is same as middle
    // element then return mid
    if (arr[mid] == x)
        return mid;

    // If x is greater than arr[mid],
    // then either arr[mid + 1] is
    // ceiling of x or ceiling lies
    // in arr[mid+1...high]
    else if (arr[mid] < x) {
        if (mid + 1 <= high && x <= arr[mid + 1])
            return mid + 1;
        else
            return ceilSearch(arr, mid + 1, high, x);
    }

    // If x is smaller than arr[mid],
    // then either arr[mid] is ceiling
    // of x  or ceiling lies in
    // arr[mid-1...high]
    else {
        if (mid - 1 >= low && x > arr[mid - 1])
            return mid;
        else
            return ceilSearch(arr, low, mid - 1, x);
    }
```

```
    }

    // Driver code
    public static void Main()
    {
        int[] arr = { 1, 2, 8, 10, 10, 12, 19 };
        int n = arr.Length;
        int x = 8;
        int index = ceilSearch(arr, 0, n - 1, x);
        if (index == -1)
            Console.Write("Ceiling of " + x +
                      " doesn't exist in array");
        else
            Console.Write("ceiling of " + x +
                           " is " + arr[index]);
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP Program for Ceiling in
// a sorted array

// Function to get index of ceiling
// of x in arr[low..high]
function ceilSearch($arr, $low,
                    $high, $x)
{
    $mid;

    /* If x is smaller than or
       equal to the first element,
       then return the first element */
    if($x <= $arr[$low])
        return $low;

    /* If x is greater than the
       last element, then return
       -1 */
    if($x > $arr[$high])
        return -1;

    /* get the index of middle
       element of arr[low..high] */
    // low + (high - low)/2
```

```
    $mid = ($low + $high)/2;

    /* If x is same as middle element,
       then return mid */
    if($arr[$mid] == $x)
        return $mid;

    /* If x is greater than arr[mid],
       then either arr[mid + 1]    is
       ceiling of x or ceiling lies
       in arr[mid+1...high] */
    else if($arr[$mid] < $x)
    {
        if($mid + 1 <= $high &&
           $x <= $arr[$mid + 1])
            return $mid + 1;
        else
            return ceilSearch($arr, $mid + 1,
                              $high, $x);
    }

    /* If x is smaller than arr[mid],
       then either arr[mid] is ceiling
       of x or ceiling lies in
       arr[mid-1...high] */
    else
    {
        if($mid - 1 >= $low &&
           $x > $arr[$mid - 1])
            return $mid;
        else
         return ceilSearch($arr, $low,
                           $mid - 1, $x);
    }
}

// Driver Code
$arr = array(1, 2, 8, 10, 10, 12, 19);
$n = sizeof($arr);
$x = 20;
$index = ceilSearch($arr, 0, $n - 1, $x);
if($index == -1)
    echo("Ceiling of $x doesn't exist in array ");
else
    echo("ceiling of $x is");
    echo(isset($arr[$index]));

// This code is contributed by nitin mittal.
```

```
?>
```

Output :

```
Ceiling of 20 doesn't exist in array
```

Time Complexity: O(Logn)

Related Articles:
Floor in a Sorted Array
Find floor and ceil in an unsorted array

Improved By : jit_t, nitin mittal

Source

https://www.geeksforgeeks.org/ceiling-in-a-sorted-array/

Chapter 15

Check if a key is present in every segment of size k in an array

Check if a key is present in every segment of size k in an array - GeeksforGeeks

Given an array arr[] and size of array is n and one another key x, and give you a segment size k. The task is to find that the key x present in every segment of size k in arr[].

Examples:

Input :
arr[] = { 3, 5, 2, 4, 9, 3, 1, 7, 3, 11, 12, 3}
x = 3
k = 3
Output : Yes
There are 4 non-overlapping segments of size k in the array, {3, 5, 2}, {4, 9, 3}, {1, 7, 3} and {11, 12, 3}. 3 is present all segments.

Input :
arr[] = { 21, 23, 56, 65, 34, 54, 76, 32, 23, 45, 21, 23, 25}
x = 23
k = 5
Output : Yes
There are three segments and last segment is not full {21, 23, 56, 65, 34}, {54, 76, 32, 23, 45} and {21, 23, 25}.
23 is present all window.

Input :arr[] = { 5, 8, 7, 12, 14, 3, 9}
x = 8
k = 2
Output : No

The idea is simple, we consider every segment of size k and check if x is present in the window or not. We need to carefully handle the last segment.

Below is the implementation of the above approach:

C++

```
 // C++ code to find the  every segment size of
// array have a search key x
#include <bits/stdc++.h>
using namespace std;

bool findxinkindowSize(int arr[], int x, int k, int n)
{
    int i;
    for (i = 0; i < n; i = i + k) {

        // Search x in segment starting
        // from index i.
        int j;
        for (j = 0; j < k; j++)
            if (arr[i + j] == x)
                break;

        // If loop didn't break
        if (j == k)
           return false;
    }

    // If n is a multiple of k
    if (i == n)
       return true;

    // Check in last segment if n
    // is not multiple of k.
    int j;
    for (j=i-k; j<n; j++)
      if (arr[j] == x)
          break;
    if (j == n)
       return false;

    return true;
}

// main driver
int main()
{
```

```
    int arr[] = { 3, 5, 2, 4, 9, 3, 1, 7, 3, 11, 12, 3 };
    int x = 3, k = 3;
    int n = sizeof(arr) / sizeof(arr[0]);
    if (findxinkindowSize(arr, x, k, n))
        cout << "Yes" << endl;
    else
        cout << "No" << endl;
    return 0;
}
```

Java

```
 // Java code to find the every
// segment size of array have
// a search key x
class GFG
{
static boolean findxinkindowSize(int arr[], int x,
                                 int k, int n)
{
    int i;
    for (i = 0; i < n; i = i + k)
    {

        // Search x in segment
        // starting from index i.
        int j;
        for (j = 0; j < k; j++)
            if (arr[i + j] == x)
                break;

        // If loop didn't break
        if (j == k)
        return false;
    }

    // If n is a multiple of k
    if (i == n)
    return true;

    // Check in last segment if
    // n is not multiple of k.
    int j;
    for (j = i - k; j < n; j++)
    if (arr[j] == x)
        break;
    if (j == n)
    return false;
```

```
    return true;
}

// Driver Code
public static void main(String args[])
{
    int arr[] = new int[] {3, 5, 2, 4, 9, 3,
                           1, 7, 3, 11, 12, 3};
    int x = 3, k = 3;
    int n = arr.length;
    if (findxinkindowSize(arr, x, k, n))
        System.out.println("Yes");
    else
        System.out.println("No");
}
}

// This code is contributed by Kirti_Mangal
```

Python 3

```
 # Python 3 program to find
# the every segment size of
# array have a search key x

def findxinkindowSize(arr, x, k, n) :

    i = 0
    while i < n :

        j = 0

        # Search x in segment
        # starting from index i
        while j < k :

            if arr[i + j] == x :
                break

            j += 1

        # If loop didn't break
        if j == k :
            return False

        i += k
```

```
    # If n is a multiple of k
    if i == n :
        return True

    j = i - k

    # Check in last segment if n
    # is not multiple of k.
    while j < n :
        if arr[j] == x :
            break

        j += 1

    if j == n :
        return False

    return True

# Driver Code
if __name__ == "__main__" :

    arr = [ 3, 5, 2, 4, 9, 3,
            1, 7, 3, 11, 12, 3 ]
    x, k = 3, 3
    n = len(arr)

    if (findxinkindowSize(arr, x, k, n)) :
        print("Yes")
    else :
        print("No")

# This code is contributed
# by ANKITRAI1
```

Output:

```
Yes
```

Time Complexity: O(n)

Improved By : Kirti_Mangal, ANKITRAI1

Source

https://www.geeksforgeeks.org/check-if-a-key-is-present-in-every-segment-of-size-k-in-an-array/

Chapter 16

Check if a string is suffix of another

Check if a string is suffix of another - GeeksforGeeks

Given two strings s1 and s2, check if s1 is a suffix of s2. Or in simple words we need to find whether string s2 ends with string s1.
Examples :

```
Input : s1 = "geeks" and s2 = "geeksforgeeks"
Output : Yes

Input : s1 = "world", s2 = "my first code is hello world"
Output : Yes

Input : s1 = "geeks" and s2 = "geeksforGeek"
Output : No
```

Method 1 (Writing our own code)

C++

```
 // CPP program to find if a string is
// suffix of another
#include <iostream>
#include <string>
using namespace std;

bool isSuffix(string s1, string s2)
{
    int n1 = s1.length(), n2 = s2.length();
    if (n1 > n2)
```

```
      return false;
    for (int i=0; i<n1; i++)
       if (s1[n1 - i - 1] != s2[n2 - i - 1])
           return false;
    return true;
}

int main()
{
    string s1 = "geeks", s2 = "geeksforgeeks";

    // Test case-sensitive implementation
    // of endsWith function
    bool result = isSuffix(s1, s2);

    if (result)
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

PHP

```
 <?php
// PHP program to find if a
// string is suffix of another
function isSuffix($s1, $s2)
{
    $n1 = ($s1);
    $n2 = strlen($s2);
    if ($n1 > $n2)
    return false;
    for ($i = 0; $i < $n1; $i++)
    if ($s1[$n1 - $i - 1] != $s2[$n2 - $i - 1])
        return false;
    return true;
}
// Driver Code
$s1 = "geeks";
$s2 = "geeksforgeeks";

// Test case-sensitive implementation
// of endsWith function
$result = isSuffix($s1, $s2);

if ($result)
```

```
    echo "Yes";
else
    echo "No";

// This code is contributed by m_kit
?>
```

Output:

```
Yes
```

Method 2 (Using boost library in C++)
Since std::string class does not provide any endWith() function which a string ends with another string so we will be using Boost Library. Make sure to include #include boost/algorithm/string.hpp and #include string to run the code fine.

C++

```
 // CPP program to find if a string is
// suffix of another
#include <boost/algorithm/string.hpp>
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s1 = "geeks", s2 = "geeksforgeeks";

    // Test case-sensitive implementation
    // of endsWith function
    bool result = boost::algorithm::ends_with(s2, s1);

    if (result)
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Output :

```
Yes
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/check-if-a-string-is-suffix-of-another/

Chapter 17

Check if all occurrences of a character appear together

Check if all occurrences of a character appear together - GeeksforGeeks

Given a string **s** and a character **c**, find if all occurrences of **c** appear together in **s** or not. If the character **c** does not appear in the string at all, the answer is true.

Examples

```
Input: s = "1110000323", c = '1'
Output: Yes
All occurrences of '1' appear together in
"1110000323"

Input: s  = "3231131", c = '1'
Output: No
All occurrences of 1 are not together

Input: s  = "abcabc", c = 'c'
Output: No
All occurrences of 'c' are not together

Input: s  = "ababcc", c = 'c'
Output: Yes
All occurrences of 'c' are together
```

The idea is to traverse given string, as soon as we find an occurrence of c, we keep traversing until we find a character which is not c. We also set a flag to indicate that one more occurrences of c are seen. If we see c again and flag is set, then we return false.

C++

```
 // CPP program to find if all occurrences
// of a character appear together in a string.
#include <iostream>
#include <string>
using namespace std;

bool checkIfAllTogether(string s, char c)
{
    // To indicate if one or more occurrences
    // of 'c' are seen or not.
    bool oneSeen = false;

    // Traverse given string
    int i = 0, n = s.length();
    while (i < n) {

        // If current character is same as c,
        // we first check if c is already seen.
        if (s[i] == c) {
            if (oneSeen == true)
                return false;

            // If this is very first appearance of c,
            // we traverse all consecutive occurrences.
            while (i < n && s[i] == c)
                i++;

            // To indicate that character is seen  once.
            oneSeen = true;
        }

        else
            i++;
    }
    return true;
}

// Driver program
int main()
{
    string s = "110029";
    if (checkIfAllTogether(s, '1'))
        cout << "Yes" << endl;
    else
        cout << "No" << endl;
    return 0;
}
```

Java

```
 // Java program to find if all
// occurrences of a character
// appear together in a string.
import java.io.*;

class GFG {

static boolean checkIfAllTogether(String s,
                                  char c)
    {

        // To indicate if one or more
        // occurrences of 'c' are seen
        // or not.
        boolean oneSeen = false;

        // Traverse given string
        int i = 0, n = s.length();
        while (i < n)
        {

            // If current character is
            // same as c, we first check
            // if c is already seen.
            if (s.charAt(i) == c)
            {
                if (oneSeen == true)
                    return false;

                // If this is very first
                // appearance of c, we
                // traverse all consecutive
                // occurrences.
                while (i < n && s.charAt(i) == c)
                    i++;

                // To indicate that character
                // is seen once.
                oneSeen = true;
            }

            else
                i++;
        }

        return true;
```

```
    }

    // Driver Code
    public static void main(String[] args)
    {

        String s = "110029";

        if (checkIfAllTogether(s, '1'))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

// This code is contributed by Sam007.
```

Python3

```
 # Python program to find
# if all occurrences
# of a character appear
# together in a string.

# function to find
# if all occurrences
# of a character appear
# together in a string.
def checkIfAllTogether(s, c) :

    # To indicate if one or
    # more occurrences of
    # 'c' are seen or not.
    oneSeen = False

    # Traverse given string
    i = 0
    n = len(s)
    while (i < n) :
        # If current character
        # is same as c,
        # we first check
        # if c is already seen.
        if (s[i] == c) :
            if (oneSeen == True) :
                return False
            # If this is very first
            # appearance of c,
```

```
            # we traverse all
            # consecutive occurrences.
            while (i < n and s[i] == c) :
                i = i + 1
            # To indicate that character
            # is seen once.
            oneSeen = True

        else :
            i = i + 1

    return True


# Driver Code
s = "110029";
if (checkIfAllTogether(s, '1')) :
    print ("Yes\n")
else :
    print ("No\n")

# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
 // C# program to find if all occurrences
// of a character appear together in a
// string.
using System;

public class GFG {

    static bool checkIfAllTogether(string s,
                                   char c)
    {

        // To indicate if one or more
        // occurrences of 'c' are seen
        // or not.
        bool oneSeen = false;

        // Traverse given string
        int i = 0, n = s.Length;
        while (i < n) {

            // If current character is
            // same as c, we first check
```

```
            // if c is already seen.
            if (s[i] == c) {
                if (oneSeen == true)
                    return false;

                // If this is very first
                // appearance of c, we
                // traverse all consecutive
                // occurrences.
                while (i < n && s[i] == c)
                    i++;

                // To indicate that character
                // is seen once.
                oneSeen = true;
            }

            else
                i++;
        }

        return true;
    }

    // Driver code
    public static void Main()
    {
        string s = "110029";

        if (checkIfAllTogether(s, '1'))
            Console.Write( "Yes" );
        else
            Console.Write( "No" );
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to find
// if all occurrences
// of a character appear
// together in a string.

// function to find
// if all occurrences
```

```
// of a character appear
// together in a string.
function checkIfAllTogether($s, $c)
{

    // To indicate if one or
    // more occurrences of
    // 'c' are seen or not.
    $oneSeen = false;

    // Traverse given string
    $i = 0; $n = strlen($s);
    while ($i < $n)
    {

        // If current character
        // is same as c,
        // we first check
        // if c is already seen.
        if ($s[$i] == $c)
        {
            if ($oneSeen == true)
                return false;

            // If this is very first
            // appearance of c,
            // we traverse all
            // consecutive occurrences.
            while ($i < $n && $s[$i] == $c)
                $i++;

            // To indicate that character
            // is seen once.
            $oneSeen = true;
        }

        else
            $i++;
    }
    return true;
}

// Driver Code
$s = "110029";
if (checkIfAllTogether($s, '1'))
    echo("Yes\n");
else
    echo("No\n");
```

```
// This code is contributed by Ajit.
?>
```

Output:

```
Yes
```

The complexity of above program is O(n).

Improved By : Sam007, jit_t, manishshaw1

Source

https://www.geeksforgeeks.org/check-occurrences-character-appear-together/

Chapter 18

Check if an array contains all elements of a given range

Check if an array contains all elements of a given range - GeeksforGeeks

An array containing positive elements is given. 'A' and 'B' are two numbers defining a range. Write a function to check if the array contains all elements in the given range.

Examples :

```
Input : arr[] = {1 4 5 2 7 8 3}
           A : 2, B : 5
Output : Yes

Input : arr[] = {1 4 5 2 7 8 3}
          A : 2, B : 6
Output : No
```

Method 1 : (Intuitive)
The most intuitive approach is to sort the array and check from the element greater than 'A' to the element greater than 'B'. If these elements are in continuous order, all elements in the range exists in the array.

Time complexity : O(n log n)
Auxiliary space : O(1)

Method 2 : (Hashing)
We can maintain a count array or a hash table which stores the count of all numbers in the array that are in the range A…B. Then we can simply check if every number occurred at least once.

Time complexity : O(n)
Auxiliary space : O(max_element)

Method 3 : (Best)
Do a linear traversal of the array. If an element is found such that |arr[i]| >= A and |arr[i]|

C++

```
 #include <iostream>
using namespace std;

// Function to check the array for elements in
// given range
bool check_elements(int arr[], int n, int A, int B)
{
    // Range is the no. of elements that are
    // to be checked
    int range = B - A;

    // Traversing the array
    for (int i = 0; i < n; i++) {

        // If an element is in range
        if (abs(arr[i]) >= A && abs(arr[i]) <= B) {

            // Negating at index 'element - A'
            int z = abs(arr[i]) - A;
            if (arr[z] > 0) {
                arr[z] = arr[z] * -1;
            }
        }
    }

    // Checking whether elements in range 0-range
    // are negative
    int count=0;
    for (int i = 0; i <= range && i<n; i++) {

        // Element from range is missing from array
        if (arr[i] > 0)
            return false;
        else
            count++;
    }
     if(count!= (range+1))
        return false;
    // All range elements are present
    return true;
}

// Driver code
```

```
int main()
{
    // Defining Array and size
    int arr[] = { 1, 4, 5, 2, 7, 8, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // A is lower limit and B is the upper limit
    // of range
    int A = 2, B = 5;

    // True denotes all elements were present
    if (check_elements(arr, n, A, B))
        cout << "Yes";

    // False denotes any element was not present
    else
        cout << "No";

    return 0;
}
```

Java

```
 // JAVA Code for Check if an array contains
// all elements of a given range
import java.util.*;

class GFG {

    // Function to check the array for elements in
    // given range
     public static boolean check_elements(int arr[], int n,
                                          int A, int B)
    {
        // Range is the no. of elements that are
        // to be checked
        int range = B - A;

        // Traversing the array
        for (int i = 0; i < n; i++) {

            // If an element is in range
            if (Math.abs(arr[i]) >= A &&
                        Math.abs(arr[i]) <= B) {


                int z = Math.abs(arr[i]) - A;
                if (arr[z] > 0) {
```

```
                    arr[z] = arr[z] * -1;
                }
            }
        }

        // Checking whether elements in range 0-range
        // are negative
        int count=0;

        for (int i = 0; i <= range && i<n; i++) {

            // Element from range is missing from array
            if (arr[i] > 0)
                return false;
            else
                count++;
        }

        if(count!= (range+1))
            return false;

        // All range elements are present
        return true;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        // Defining Array and size
        int arr[] = { 1, 4, 5, 2, 7, 8, 3 };
        int n = arr.length;

        // A is lower limit and B is the upper limit
        // of range
        int A = 2, B = 5;

        // True denotes all elements were present
        if (check_elements(arr, n, A, B))
            System.out.println("Yes");

        // False denotes any element was not present
        else
            System.out.println("No");
    }
}
// This code is contributed by Arnav Kr. Mandal.
```

C#

```
 // C# Code for Check if an array contains
// all elements of a given range
using System;

class GFG {

    // Function to check the array for
    // elements in given range
    public static bool check_elements(int []arr, int n,
                                      int A, int B)
    {
        // Range is the no. of elements
        // that are to be checked
        int range = B - A;

        // Traversing the array
        for (int i = 0; i < n; i++)
        {

            // If an element is in range
            if (Math.Abs(arr[i]) >= A &&
                Math.Abs(arr[i]) <= B)
                {
                    int z = Math.Abs(arr[i]) - A;
                    if (arr[z] > 0)
                    {
                      arr[z] = arr[z] * - 1;
                    }
                }
        }

        // Checking whether elements in
        // range 0-range are negative
        int count=0;

        for (int i = 0; i <= range
             && i < n; i++)
        {

            // Element from range is
            // missing from array
            if (arr[i] > 0)
                return false;
            else
                count++;
        }

        if(count != (range + 1))
```

```
            return false;

        // All range elements are present
        return true;
    }

    // Driver Code
    public static void Main(String []args)
    {
        // Defining Array and size
        int []arr = {1, 4, 5, 2, 7, 8, 3};
        int n = arr.Length;

        // A is lower limit and B is
        // the upper limit of range
        int A = 2, B = 5;

        // True denotes all elements were present
        if (check_elements(arr, n, A, B))
         Console.WriteLine("Yes");

        // False denotes any element was not present
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// Function to check the
// array for elements in
// given range
function check_elements($arr, $n,
                        $A, $B)
{
    // Range is the no. of
    // elements that are to
    // be checked
    $range = $B - $A;

    // Traversing the array
    for ($i = 0; $i < $n; $i++)
    {

        // If an element is in range
```

```
        if (abs($arr[$i]) >= $A &&
            abs($arr[$i]) <= $B)
        {

            // Negating at index
            // 'element - A'
            $z = abs($arr[$i]) - $A;
            if ($arr[$z] > 0)
            {
                $arr[$z] = $arr[$z] * -1;
            }
        }
    }

    // Checking whether elements
    // in range 0-range are negative
    $count = 0;
    for ($i = 0; $i <= $range &&
                 $i< $n; $i++)
    {

        // Element from range is
        // missing from array
        if ($arr[$i] > 0)
            return -1;
        else
            $count++;
    }
    if($count!= ($range + 1))
        return -1;
    // All range elements
    // are present
    return true;
}

// Driver code

// Defining Array and size
$arr = array(1, 4, 5, 2,
             7, 8, 3);
$n = sizeof($arr);

// A is lower limit and
// B is the upper limit
// of range
$A = 2; $B = 5;

// True denotes all
```

```
// elements were present
if ((check_elements($arr, $n,
                    $A, $B)) == true)

    echo "Yes";

// False denotes any
// element was not present
else
    echo "No";

// This code is contributed by aj_36
?>
```

Output :

```
Yes
```

Time complexity : O(n)
Auxiliary space : O(1)

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/check-if-an-array-contains-all-elements-of-a-given-range/

Chapter 19

Check if an array has a majority element

Check if an array has a majority element - GeeksforGeeks

Given an array, the task is to find if the input array contains a majority element or not. An element is

Examples:

```
Input : arr[] = {2, 3, 9, 2, 2}
Output : Yes
A majority element 2 is present in arr[]

Input  : arr[] = {1, 8, 9, 2, 5}
Output : No
```

A **simple solution** is to traverse through array. For every element, count its occurrences. If count of occurrence of any element becomes n/2, we return true.

An **efficient solution** is to use hashing. We count occurrences of all elements. If count becomes n/2 or more return true.

Below is the implementation of the approach.

C++

```
 // Hashing based C++ program to find if there
// is a majority element in input array.
#include <bits/stdc++.h>
using namespace std;
```

```
// Returns true if there is a majority element
// in a[]
bool isMajority(int a[], int n)
{
    // Insert all elements in a hash table
    unordered_map<int, int> mp;
    for (int i = 0; i < n; i++)
        mp[a[i]]++;

    // Check if frequency of any element is
    // n/2 or more.
    for (auto x : mp)
      if (x.second >= n/2)
          return true;
    return false;
}

// Driver code
int main()
{
    int a[] = { 2, 3, 9, 2, 2 };
    int n = sizeof(a) / sizeof(a[0]);
    if (isMajority(a, n))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
 // Hashing based Java program
// to find if there is a
// majority element in input array.
import java.util.*;
import java.lang.*;
import java.io.*;

class Gfg
{
    // Returns true if there is a
    // majority element in a[]
    static boolean isMajority(int a[], int n)
    {
        // Insert all elements
        // in a hash table
        HashMap <Integer,Integer> mp = new
                            HashMap<Integer,Integer>();
```

```
        for (int i = 0; i < n; i++)

            if (mp.containsKey(a[i]))
                mp.put(a[i], mp.get(a[i]) + 1);

            else mp.put(a[i] , 1);

        // Check if frequency of any
        // element is n/2 or more.
        for (Map.Entry<Integer, Integer> x : mp.entrySet())

            if (x.getValue() >= n/2)
                return true;
        return false;
    }

    // Driver code
    public static void main (String[] args)
    {
        int a[] = { 2, 3, 9, 2, 2 };
        int n = a.length;

        if (isMajority(a, n))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

// This code is contributed by Ansu Kumari
```

Python3

```
 # Hashing based Python
# program to find if
# there is a majority
# element in input array.

# Returns true if there
# is a majority element
# in a[]
def isMajority(a):

    # Insert all elements
    # in a hash table
    mp = {}
```

```
    for i in a:
        if i in mp: mp[i] += 1
        else: mp[i] = 1

    # Check if frequency
    # of any element is
    # n/2 or more.
    for x in mp:
        if mp[x] >= len(a)//2:
            return True
    return False

# Driver code
a = [ 2, 3, 9, 2, 2 ]

print("Yes" if isMajority(a) else "No")

#This code is contributed by Ansu Kumari
```

Output:1

```
Yes
```

Improved By : bhatnagarm

Source

https://www.geeksforgeeks.org/check-array-majority-element/

Chapter 20

Check if reversing a sub array make the array sorted

Check if reversing a sub array make the array sorted - GeeksforGeeks

Given an array of distinct **n** integers. The task is to check whether reversing one sub-array make the array sorted or not. If the array is already sorted or by reversing a subarray once make it sorted, print "Yes", else print "No".

Examples:

```
Input : arr [] = {1, 2, 5, 4, 3}
Output : Yes
By reversing the subarray {5, 4, 3},
the array will be sorted.

Input : arr [] = { 1, 2, 4, 5, 3 }
Output : No
```

Method 1 (Simple : $O(n^2)$)
A simple solution is to consider every subarray one by one. Try reversing every subarray and check if reversing the subarray makes the whole array sorted. If yes, return true. If reversing any subarray doesn't make the array sorted, then return false.

Method 2 (Sorting : O(nlogn)):
The idea is to compare the given array with the sorted array. Make a copy of the given array and sort it. Now, find the first index and last index which do not match with sorted array. If no such indices are found, print "Yes". Else check if the elements between the indices are in decreasing order.

```
// C++ program to check whether reversing a
```

```
// sub array make the array sorted or not
#include<bits/stdc++.h>
using namespace std;

// Return true, if reversing the subarray will
// sort the array, else return false.
bool checkReverse(int arr[], int n)
{
    // Copying the array.
    int temp[n];
    for (int i = 0; i < n; i++)
        temp[i] = arr[i];

    // Sort the copied array.
    sort(temp, temp + n);

    // Finding the first mismatch.
    int front;
    for (front = 0; front < n; front++)
        if (temp[front] != arr[front])
            break;

    // Finding the last mismatch.
    int back;
    for (back = n - 1; back >= 0; back--)
        if (temp[back] != arr[back])
            break;

    // If whole array is sorted
    if (front >= back)
        return true;

    // Checking subarray is decreasing or not.
    do
    {
        front++;
        if (arr[front - 1] < arr[front])
            return false;
    } while (front != back);

    return true;
}

// Driven Program
int main()
{
    int arr[] = { 1, 2, 5, 4, 3 };
    int n = sizeof(arr)/sizeof(arr[0]);
```

```
    checkReverse(arr, n)? (cout << "Yes" << endl):
                          (cout << "No" << endl);
    return 0;
}
```

Output:

```
Yes
```

Time Complexity: O(nlogn).

Method 3 (Linear : O(n)):
Observe, answer will be "Yes" when the array is sorted or when the array consist of three parts. First part is increasing subarray, then decreasing subarray and then again increasing subarray. So, we need to check that array contain increasing elements then some decreasing elements and then increasing elements. In all other case, answer will be "No".

Below is the C++ implementation of this approach:

```
 // C++ program to check whether reversing a sub array
// make the array sorted or not
#include<bits/stdc++.h>
using namespace std;

// Return true, if reversing the subarray will sort t
// he array, else return false.
bool checkReverse(int arr[], int n)
{
    if (n == 1)
        return true;

    // Find first increasing part
    int i;
    for (i=1; i < n && arr[i-1] < arr[i]; i++);
    if (i == n)
        return true;

    // Find reversed part
    int j = i;
    while (arr[j] < arr[j-1])
    {
        if (i > 1 && arr[j] < arr[i-2])
            return false;
        j++;
    }
```

```
    if (j == n)
        return true;

    // Find last increasing part
    int k = j;

    // To handle cases like {1,2,3,4,20,9,16,17}
    if (arr[k] < arr[i-1])
       return false;

    while (k > 1 && k < n)
    {
        if (arr[k] < arr[k-1])
            return false;
        k++;
    }
    return true;
}

// Driven Program
int main()
{
    int arr[] = {1, 3, 4, 10, 9, 8};
    int n = sizeof(arr)/sizeof(arr[0]);
    checkReverse(arr, n)? cout << "Yes" : cout << "No";
    return 0;
}
```

Output:

```
Yes
```

Time Complexity: O(n).

Source

https://www.geeksforgeeks.org/check-reversing-sub-array-make-array-sorted/

Chapter 21

Check if there exist two elements in an array whose sum is equal to the sum of rest of the array

Check if there exist two elements in an array whose sum is equal to the sum of rest of the array - GeeksforGeeks

We have an array of integers and we have to find two such elements in the array such that sum of these two elements is equal to the sum of rest of elements in array.

Examples:

```
Input  : arr[] = {2, 11, 5, 1, 4, 7}
Output : Elements are 4 and 11
Note that 4 + 11 = 2 + 5 + 1 + 7

Input  : arr[] = {2, 4, 2, 1, 11, 15}
Output : Elements do not exist
```

A **simple solution** is to consider every pair one by one, find its sum and compare the sum with sum of rest of the elements. If we find a pair whose sum is equal to rest of elements, we print the pair and return true. Time complexity of this solution is $O(n^3)$

An **efficient solution** is to find sum of all array elements. Let this sum be "sum". Now the task reduces to finding a pair with sum equals to sum/2.
Another optimization is, a pair can exist only if the sum of whole array is even because we are basically dividing it into two parts with equal sum.

1- Find the sum of whole array. Let this sum be "sum"
2- If sum is odd, return false.

3- Find a pair with sum equals to "sum/2" using hashing based method discussed here as method 2. If a pair is found, print it and return true.
4- If no pair exists, return false.

Below is C++ implementation of above steps.

```
 // C++ program to find whether two elements exist
// whose sum is equal to sum of rest of the elements.
#include<bits/stdc++.h>
using namespace std;

// Function to check whether two elements exist
// whose sum is equal to sum of rest of the elements.
bool checkPair(int arr[],int n)
{
    // Find sum of whole array
    int sum = 0;
    for (int i=0; i<n; i++)
        sum += arr[i];

    // If sum of array is not even than we can not
    // divide it into two part
    if (sum%2 != 0)
        return false;

    sum = sum/2;

    // For each element arr[i], see if there is
    // another element with vaalue sum - arr[i]
    unordered_set<int> s;
    for (int i=0; i<n; i++)
    {
        int val = sum-arr[i];

        // If element exist than return the pair
        if (s.find(val) != s.end())
        {
            printf("Pair elements are %d and %d\n",
                                arr[i], val);
            return true;
        }

        s.insert(arr[i]);
    }

    return false;
}

// Driver program.
```

```
int main()
{
    int arr[] = {2, 11, 5, 1, 4, 7};
    int n = sizeof(arr)/sizeof(arr[0]);
    if (checkPair(arr, n) == false)
       printf("No pair found");
    return 0;
}
```

Output:

```
Pair elements are 4 and 11
```

Time complexity : O(n). unordered_set is implemented using hashing. Time complexity hash search and insert is assumed as O(1) here.

Source

https://www.geeksforgeeks.org/check-exist-two-elements-array-whose-sum-equal-sum-rest-array/

Chapter 22

Closest greater element for every array element from another array

Closest greater element for every array element from another array - GeeksforGeeks

Given two arrays a[] and b[], we need to build an array c[] such that every element c[i] of c[] contains a value from a[] which is greater than b[i] and is closest to b[i]. If a[] has no greater element than b[i], then value of c[i] is -1. All arrays are of same size.

Examples:

```
Input : a[] = [ 2, 6, 5, 7, 0]
        b[] = [1, 3, 2, 5, 8]
Output : c[] = [2, 5, 5, 7, -1]

Input : a[] = [ 2, 6, 5, 7, 0]
        b[] = [0, 2, 3, 5, 1]
Output : c[] = [2, 5, 5, 6, 2]
```

Naive Approach : For each element in b[], we traverse the whole of a[] and try to find the closest greater element, and save the result for each search. This will cost time complexity of O(n^2).

Efficient Approach : Sort the array a[], and for each b[i], apply binary search in sorted array a[]. For this method our time complexity will be O(nlogn).
Note: For closest greater element we can use upper_bound().

```
 // CPP to find result from target array
// for closest element
#include <bits/stdc++.h>
```

```
using namespace std;

// Function for printing resultant array
void closestResult(int a[], int b[], int n)
{
    // change arr[] to vector
    vector<int> vect(a, a + n);

    // sort vector for ease
    sort(vect.begin(), vect.end());

    // iterator for upper_bound
    vector<int>::iterator up;

    // vector for result
    vector<int> c;

    // calculate resultant array
    for (int i = 0; i < n; i++) {

        // check upper bound element
        up = upper_bound(vect.begin(), vect.end(), b[i]);

        // if no element found push -1
        if (up == vect.end())
            c.push_back(-1);

        // Else push the element
        else
            c.push_back(*up); // add to resultant
    }

    cout << "Result = ";
    for (auto it = c.begin(); it != c.end(); it++)
        cout << *it << " ";
}

// driver program
int main()
{
    int a[] = { 2, 5, 6, 1, 8, 9 };
    int b[] = { 2, 1, 0, 5, 4, 9 };
    int n = sizeof(a) / sizeof(a[0]);
    closestResult(a, b, n);
    return 0;
}
```

Output:

```
Result = 5 2 1 6 5 -1
```

Source

https://www.geeksforgeeks.org/closest-greater-element-every-array-element-another-array/

Chapter 23

Closest numbers from a list of unsorted integers

Closest numbers from a list of unsorted integers - GeeksforGeeks

Given a list of distinct unsorted integers, find the pair of elements that have the smallest absolute difference between them? If there are multiple pairs, find them all.

Examples:

```
Input : arr[] = {10, 50, 12, 100}
Output : (10, 12)
The closest elements are 10 and 12

Input : arr[] = {5, 4, 3, 2}
Output : (2, 3), (3, 4), (4, 5)
```

This problem is mainly an extension ofFind minimum difference between any two elements.

1. Sort the given array.
2. Find minimum difference of all pairs in linear time in sorted array.
3. Traverse sorted array one more time to print all pairs with minimum difference obtained in step 2.

C++

```
 // CPP program to find minimum difference
// an unsorted array.
#include<bits/stdc++.h>
using namespace std;
```

```
// Returns minimum difference between any
// two pair in arr[0..n-1]
void printMinDiffPairs(int arr[], int n)
{
   if (n <= 1)
      return;

   // Sort array elements
   sort(arr, arr+n);

   // Compare differences of adjacent
   // pairs to find the minimum difference.
   int minDiff = arr[1] - arr[0];
   for (int i = 2 ; i < n ; i++)
      minDiff = min(minDiff, arr[i] - arr[i-1]);

   // Traverse array again and print all pairs
   // with difference as minDiff.
   for (int i = 1; i < n; i++)
      if ((arr[i] - arr[i-1]) == minDiff)
         cout << "(" << arr[i-1] << ", "
              << arr[i] << "), ";
}

// Driver code
int main()
{
    int arr[] = {5, 3, 2, 4, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    printMinDiffPairs(arr, n);
    return 0;
}
```

Java

```
 // Java program to find minimum
// difference an unsorted array.
import java.util.*;

class GFG
{

  // Returns minimum difference between
  // any two pair in arr[0..n-1]
  static void printMinDiffPairs(int arr[], int n)
  {
    if (n <= 1)
      return;
```

```
    // Sort array elements
    Arrays.sort(arr);

    // Compare differences of adjacent
    // pairs to find the minimum difference.
    int minDiff = arr[1] - arr[0];
    for (int i = 2; i < n; i++)
      minDiff = Math.min(minDiff, arr[i] - arr[i-1]);

    // Traverse array again and print all pairs
    // with difference as minDiff.
    for ( int i = 1; i < n; i++)
     {
        if ((arr[i] - arr[i-1]) == minDiff)
        {
           System.out.print("(" + arr[i-1] + ", "
                           + arr[i] + ")," );
        }
     }
  }

  // Driver code
  public static void main (String[] args)
  {
    int arr[] = {5, 3, 2, 4, 1};
    int n = arr.length;
    printMinDiffPairs(arr, n);
  }
}

// This code is contributed by Ansu Kumari
```

Python3

```
 # Python3 program to find minimum
# difference in an unsorted array.

# Returns minimum difference between
# any two pair in arr[0..n-1]
def printMinDiffPairs(arr , n):
    if n <= 1: return

    # Sort array elements
    arr.sort()

    # Compare differences of adjacent
    # pairs to find the minimum difference.
```

```
    minDiff = arr[1] - arr[0]
    for i in range(2 , n):
        minDiff = min(minDiff, arr[i] - arr[i-1])

    # Traverse array again and print all
    # pairs with difference as minDiff.
    for i in range(1 , n):
        if (arr[i] - arr[i-1]) == minDiff:
            print( "(" + str(arr[i-1]) + ", "
                 + str(arr[i]) + "), ", end = '')

# Driver code
arr = [5, 3, 2, 4, 1]
n = len(arr)
printMinDiffPairs(arr , n)

# This code is contributed by Ansu Kumari
```

C#

```
 // C# program to find minimum
// difference an unsorted array.
using System;

class GFG
{

// Returns minimum difference between
// any two pair in arr[0..n-1]
static void printMinDiffPairs(int []arr, int n)
{
    if (n <= 1)
    return;

    // Sort array elements
    Array.Sort(arr);

    // Compare differences of adjacent
    // pairs to find the minimum difference.
    int minDiff = arr[1] - arr[0];
    for (int i = 2; i < n; i++)
    minDiff = Math.Min(minDiff, arr[i] - arr[i-1]);

    // Traverse array again and print all pairs
    // with difference as minDiff.
    for ( int i = 1; i < n; i++)
    {
        if ((arr[i] - arr[i-1]) == minDiff)
```

```
        {
        Console.Write(" (" + arr[i-1] + ", "
                          + arr[i] + "), " );
        }
    }
}

// Driver code
public static void Main ()
{
    int []arr = {5, 3, 2, 4, 1};
    int n = arr.Length;
    printMinDiffPairs(arr, n);
}
}

// This code is contributed by vt_m
```

PHP

```
 <?php
//PHP program to find minimum difference
// an unsorted array.

// Returns minimum difference between any
// two pair in arr[0..n-1]
function printMinDiffPairs($arr, $n)
{
    if ($n <= 1)
        return;

    // Sort array elements
    sort($arr);

    // Compare differences of adjacent
    // pairs to find the minimum
    // difference.
    $minDiff = $arr[1] - $arr[0];

    for ($i = 2 ; $i < $n ; $i++)
        $minDiff = min($minDiff, $arr[$i]
                            - $arr[$i-1]);

    // Traverse array again and print all
    // pairs with difference as minDiff.
    for ($i = 1; $i < $n; $i++)
        if (($arr[$i] - $arr[$i-1]) ==
                                $minDiff)
```

```
            echo "(" , $arr[$i-1] , ", ",
                        $arr[$i] , "), ";
}

// Driver code
    $arr = array(5, 3, 2, 4, 1);
    $n = sizeof($arr);
    printMinDiffPairs($arr, $n);

// This code is contributed by ajit.
?>
```

Output:

```
(1, 2), (2, 3), (3, 4), (4, 5),
```

Does above program handle duplicates?
The cases like {x, x, x} are not handled by above program. For this case, the expected output (x, x), (x, x), (x, x), but above program prints (x, x), (x, x)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/closest-numbers-list-unsorted-integers/

Chapter 24

Closest product pair in an array

Closest product pair in an array - GeeksforGeeks

Given an array of non negative integers and a number x, find a pair in array whose product is closest to x.

Examples :

```
Input : arr[] = [2, 3, 5, 9]
        x = 47
Output : {5, 9}

Input : arr[] = [2, 3, 5, 9]
        x = 8
Output : {2, 5}
```

Method 1
A simple solution is to consider every pair and keep track of closest pair (absolute difference between pair product and x is minimum). Finally print the closest pair. Time complexity of this solution is $O(n^2)$

Method 2 O(n Log n)

1. Sort the array
2. Initialize a variable diff as infinite (Diff is used to store the difference between pair and x). We need to find the minimum diff.
3. Traverse the array and for each i, do the following :
    - Find the lower bound for x/arr[i] in the sub array on right of arr[i], i.e., in sub array arr[i+1..n-1]. Let it be denoted be l.
    - Find the upper bound for x/arr[i] in the sub array on right of arr[i], i.e., in sub array arr[i+1..n-1]. Let it be denoted be u.

- If min(abs((arr[i] * l) – x), abs((arr[i] * u) – x)) < diff then update diff and result

Method 3 O(n for sorted)
An efficient solution can find the pair in O(n) time. Following is detailed algorithm.

```
1) Initialize a variable diff as infinite
   (Diff is used to store the difference
   between pair and x).  We need to find
   the minimum diff.
2) Initialize two index variables l and r
   in the given sorted array.
   (a) Initialize first to the leftmost index:
        l = 0
   (b) Initialize second  the rightmost index:
        r = n-1
3) Loop while l < r.
   (a) If  abs((arr[l] * arr[r]) - x) < diff
       then update diff and result
   (b) Else if((arr[l] * arr[r]) <  x)  then
       l++
   (c) Else r--
```

Following is the implementation of above algorithm.

C++

```
 // Simple C++ program to find the pair with
// product closest to a given no.
#include <bits/stdc++.h>
using namespace std;

// Prints the pair with product closest to x
void printClosest(int arr[], int n, int x)
{
    int res_l, res_r; // To store indexes of result pair

    // Initialize left and right indexes and
    // difference between pair product and x
    int l = 0, r = n - 1, diff = INT_MAX;

    // While there are elements between l and r
    while (r > l) {

        // Check if this pair is closer than
        // the closest pair so far
        if (abs(arr[l] * arr[r] - x) < diff) {
```

```
            res_l = l;
            res_r = r;
            diff = abs(arr[l] * arr[r] - x);
        }

        // If this pair has more product,
        // move to smaller values.
        if (arr[l] * arr[r] > x)
            r--;

        else // Move to larger values
            l++;
    }

    cout << " The closest pair is "
        << arr[res_l] << " and " << arr[res_r];
}

// Driver program to test above functions
int main()
{
    int arr[] = { 2, 3, 5, 9 }, x = 8;
    int n = sizeof(arr) / sizeof(arr[0]);
    printClosest(arr, n, x);
    return 0;
}
```

Java

```
 // Simple Java program to find
// the pair with product closest
// to a given no.
import java.io.*;

class GFG
{
// Prints the pair with
// product closest to x
static void printClosest(int arr[],
                         int n, int x)
{
    // To store indexes of result pair
    int res_l = 0, res_r = 0;

    // Initialize left and right
    // indexes and difference
    // between pair product and x
    int l = 0, r = n - 1, diff = Integer.MAX_VALUE;
```

```
    // While there are
    // elements between l and r
    while (r > l)
    {

        // Check if this pair is closer
        // than the closest pair so far
        if (Math.abs(arr[l] * arr[r] - x) < diff)
        {
            res_l = l;
            res_r = r;
            diff = Math.abs(arr[l] * arr[r] - x);
        }

        // If this pair has more product,
        // move to smaller values.
        if (arr[l] * arr[r] > x)
            r--;

        // Move to larger values
        else
            l++;
    }

    System.out.print("The closest pair is ");
    System.out.print (arr[res_l] +
                          " and " +
                      arr[res_r]);
}

// Driver Code
public static void main (String[] args)
{
int arr[] = {2, 3, 5, 9};
int x = 8;
int n = arr.length;
printClosest(arr, n, x);
}
}

// This code is contributed by anuj_67.
```

Python3

```
 # Simple Python3 program to find
# the pair with product closest
# to a given no.
```

```
import sys

# Prints the pair with
# product closest to x
def printClosest(arr, n, x):

    # To store indexes
    # of result pair
    res_l = 0;
    res_r = 0;

    # Initialize left and right
    # indexes and difference
    # between pair product and x
    l = 0;
    r = n - 1;
    diff = sys.maxsize;

    # While there are elements
    # between l and r
    while (r > l):

        # Check if this pair is
        # closer than the closest
        # pair so far
        if (abs(arr[l] *
                arr[r] - x) < diff):
            res_l = l;
            res_r = r;
            diff = abs(arr[l] *
                       arr[r] - x);

        # If this pair has more
        # product, move to smaller
        # values.
        if (arr[l] * arr[r] > x):
            r = r - 1;

        # Move to larger values
        else:
            l = l + 1;

    print("The closest pair is", arr[res_l] ,
                          "and", arr[res_r]);

# Driver Code
arr = [2, 3, 5, 9];
x = 8;
```

```
n = len(arr);
printClosest(arr, n, x);

# This code is contributed
# by rahul
```

C#

```
 // Simple C# program to find
// the pair with product closest
// to a given no.
using System;

class GFG
{
// Prints the pair with
// product closest to x
static void printClosest(int []arr,
                         int n, int x)
{
    // To store indexes of result pair
    int res_l = 0, res_r = 0;

    // Initialize left and right
    // indexes and difference
    // between pair product and x
    int l = 0, r = n - 1,
        diff = int.MaxValue;

    // While there are
    // elements between l and r
    while (r > l)
    {

        // Check if this pair is closer
        // than the closest pair so far
        if (Math.Abs(arr[l] *
                     arr[r] - x) < diff)
        {
            res_l = l;
            res_r = r;
            diff = Math.Abs(arr[l] *
                            arr[r] - x);
        }

        // If this pair has more product,
        // move to smaller values.
        if (arr[l] * arr[r] > x)
```

```
            r--;

        // Move to larger values
        else
            l++;
    }

    Console.Write("The closest pair is ");
    Console.Write (arr[res_l] +
                      " and " +
                   arr[res_r]);
}

// Driver Code
public static void Main ()
{
int []arr = {2, 3, 5, 9};
int x = 8;
int n = arr.Length;
printClosest(arr, n, x);
}
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// Simple PHP program to find
// the pair with product closest
// to a given no.

// Prints the pair with
// product closest to x
function printClosest($arr, $n, $x)
{
    // To store indexes
    // of result pair
    $res_l; $res_r;

    // Initialize left and right
    // indexes and difference
    // between pair product and x
    $l = 0; $r = $n - 1; $diff = PHP_INT_MAX;

    // While there are elements
    // between l and r
    while ($r > $l)
```

```
    {

        // Check if this pair is
        // closer than the closest
        // pair so far
        if (abs($arr[$l] *
                $arr[$r] - $x) < $diff)
        {
            $res_l = $l;
            $res_r = $r;
            $diff = abs($arr[$l] *
                        $arr[$r] - $x);
        }

        // If this pair has more
        // product, move to smaller
        // values.
        if ($arr[$l] * $arr[$r] > $x)
            $r--;

        // Move to larger values
        else
            $l++;
    }

    echo " The closest pair is " ,
          $arr[$res_l] , " and " ,
                     $arr[$res_r];
}

// Driver Code
$arr = array(2, 3, 5, 9);
$x = 8;
$n = count($arr);
printClosest($arr, $n, $x);

// This code is contributed by anuj_67.
?>
```

Output :

```
The closest pair is 2 and 5
```

Improved By : vt_m, mithunkumarmnnit321

Source

https://www.geeksforgeeks.org/closest-product-pair-array/

Chapter 25

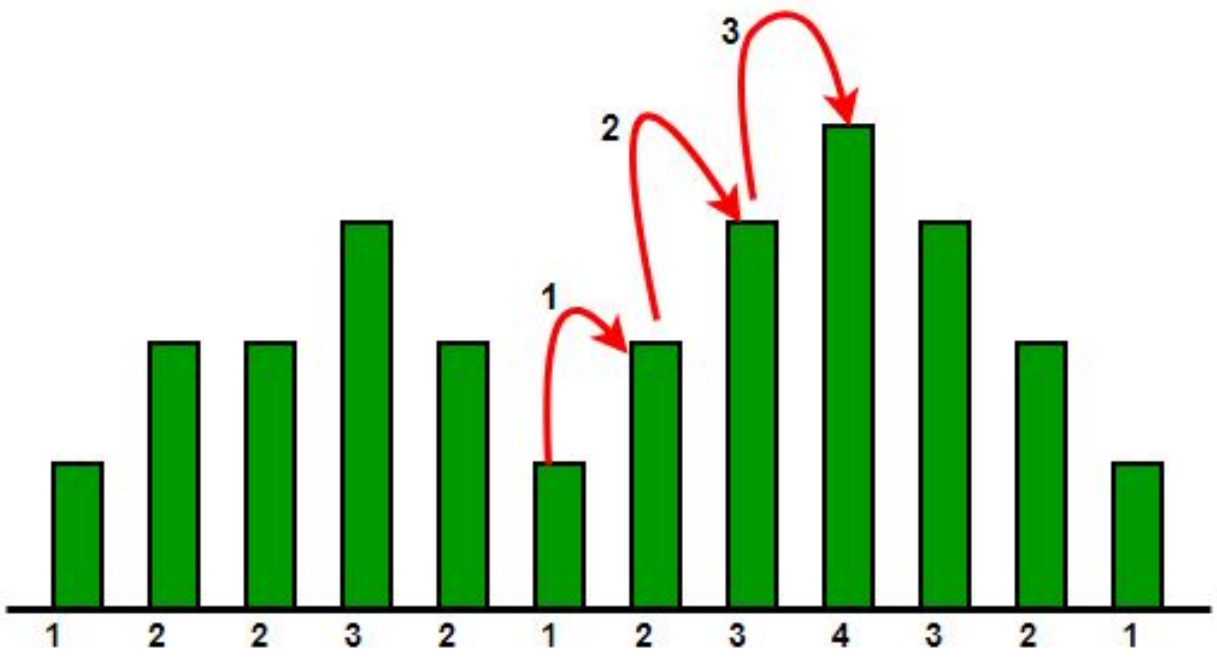
Consecutive steps to roof top

Consecutive steps to roof top - GeeksforGeeks

Given heights of consecutive buildings, find the maximum number of consecutive steps one can put forward such that he gain a increase in altitude while going from roof of one building to next adjacent one.

Examples :

```
Input : arr[] = {1, 2, 2, 3, 2}
Output : 1
Explanation :
Maximum consecutive steps from 1 to 2 OR  2 to 3.

Input : arr[] = {1, 2, 3, 4}
Output : 3
```



This problem is basically a variation of Longest increasing subarray

Approach:-

```
initialize count = 0
initialize maximum = 0
    if arr[i]>a[i-1]
then count increment
    else
maximum = max(maximum, count)

at the end maximum=max(maximum, count)
```

C++

```
 // CPP code to find maximum
// number of consecutive steps.
#include <bits/stdc++.h>
using namespace std;

// Function to count consecutive steps
int find_consecutive_steps(int arr[], int len)
{
    int count = 0;
    int maximum = 0;
```

```
    for (int index = 1; index < len; index++) {

        // count the number of consecutive
        // increasing height building
        if (arr[index] > arr[index - 1])
            count++;
        else
        {
            maximum = max(maximum, count);
            count = 0;
        }
    }

    return max(maximum, count);
}

// Driver code
int main()
{
    int arr[] = { 1, 2, 3, 4 };
    int len = sizeof(arr) / sizeof(arr[0]);

    cout << find_consecutive_steps(arr, len);
}
```

Java

```
 // Java code to find maximum
// number of consecutive steps.
import java.io.*;

class GFG {

    // Function to count consecutive steps
    static int find_consecutive_steps(int arr[],
                                      int len)
    {
        int count = 0;
        int maximum = 0;

        for (int index = 1; index < len; index++) {

            // count the number of consecutive
            // increasing height building
            if (arr[index] > arr[index - 1])
                count++;
            else
```

```
            {
                maximum = Math.max(maximum, count);
                count = 0;
            }
        }

        return Math.max(maximum, count);
    }

    // Driver code
    public static void main (String[] args) {

        int arr[] = { 1, 2, 3, 4 };
        int len = arr.length;

        System.out.println(find_consecutive_steps(arr,
                                                  len));
    }
}

// This code is contributed by Gitanjali.
```

Python3

```
 # Python3 code to find maximum
# number of consecutive steps
import math

# Function to count consecutive steps
def find_consecutive_steps(arr, len):

    count = 0; maximum = 0

    for index in range(1, len):

        # count the number of consecutive
        # increasing height building
        if (arr[index] > arr[index - 1]):
            count += 1

        else:
            maximum = max(maximum, count)
            count = 0

    return max(maximum, count)

# Driver code
arr = [ 1, 2, 3, 4 ]
```

```
len = len(arr)
print(find_consecutive_steps(arr, len))


# This code is contributed by Gitanjali.
```

C#

```
 // C# code to find maximum
// number of consecutive steps.
using System;

class GFG {

    // Function to count consecutive steps
    static int find_consecutive_steps(int []arr,
                                      int len)
    {
        int count = 0;
        int maximum = 0;

        for (int index = 1; index < len; index++) {

            // count the number of consecutive
            // increasing height building
            if (arr[index] > arr[index - 1])
                count++;
            else
            {
                maximum = Math.Max(maximum, count);
                count = 0;
            }
        }

        return Math.Max(maximum, count);
    }

    // Driver code
    public static void Main () {

        int []arr = { 1, 2, 3, 4 };
        int len = arr.Length;

        Console.WriteLine(find_consecutive_steps(arr,
                                                 len));
    }
}
```

```
// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP code to find maximum
// number of consecutive steps.


// Function to count
// consecutive steps
function find_consecutive_steps($arr,
                                $len)
{
    $count = 0;
    $maximum = 0;

    for ($index = 1; $index < $len;
                          $index++)
    {

        // count the number of consecutive
        // increasing height building
        if ($arr[$index] > $arr[$index - 1])
            $count++;
        else
        {
            $maximum = max($maximum, $count);
            $count = 0;
        }
    }

    return max($maximum, $count);
}

// Driver code
$arr = array( 1, 2, 3, 4 );
$len = count($arr);

echo find_consecutive_steps($arr, $len);

// This code is contributed by vt_m.
?>
```

Output :

```
3
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/roof-top/

Chapter 26

Cost to Balance the parantheses

Cost to Balance the parantheses - GeeksforGeeks

Parenthesis are said to be balanced when every opening brace has a closing brace like "()()" or "(())" or "((()))" etc. Incorrect balancing includes ")(" or "))((" etc. The task here is to correct the sequence of parenthesis in such a way that it is done in minimum cost. And shifting of parenthesis by over one parentheses costs 1. If the parenthesis can't be balanced then print -1.

Examples :

> Input : ()
> Output : 0
> **Explanation :** Already balanced
>
> Input :))((
> Output : 4
> **Explanation :** Firstly,) at position 1^{st} goes to the last position, costing 3, so we get)((). Then,) at position 1^{st} goes to the 2^{nd} position costing 1. So, finally we get ()(). Therefore, the total cost is 4.

Algorithm :

1. Store the braces in string.
2. Run a loop to string size to store the count of opening and closing braces.
3. Check if number of opening brace is equal to number of closing brace or not.
4. If the braces are not equal then print -1 depicting that the string cant be balanced. Else proceed further.
5. Initially, Check at 0^{th} index that whether the string contains opening brace or closing brace. If we get an opening brace then store +1 in the array at index 0, else if closing brace is present then place -1 at 0^{th} index.
6. Now run a loop from 1^{st} index to array length.

- If opening brace is present at index i then add +1 to value at previous index i.e. i-1 and store sum at index i.

- If closing brace is present at index i then add -1 to value at previous index i.e. i-1 and store sum at index i.
- If value at index i is negative i.e less than 0, then add the absolute value of array[i] into a variable(ans in below program).

Finally we get the minimum cost in variable ans.

Below is the implementation of above approach :

C++

```
 // CPP code to calculate the minimum cost
// to make the given parentheses balanced
#include <bits/stdc++.h>
using namespace std;

int costToBalance(string s)
{
    if (s.length() == 0)
        cout << 0 << endl;

    // To store absolute count of
    // balanced and unbalanced parenthesis
    int ans = 0;

    // o(open bracket) stores count of '(' and
    // c(close bracket) stores count of ')'
    int o = 0, c = 0;

    for (int i = 0; i < s.length(); i++) {
        if (s[i] == '(')
            o++;
        if (s[i] == ')')
            c++;
    }

    if (o != c)
        return -1;

    int a[s.size()];
    if (s[0] == '(')
        a[0] = 1;
    else
        a[0] = -1;

    if (a[0] < 0)
        ans += abs(a[0]);
```

```
    for (int i = 1; i < s.length(); i++) {
        if (s[i] == '(')
            a[i] = a[i - 1] + 1;
        else
            a[i] = a[i - 1] - 1;
        if (a[i] < 0)
            ans += abs(a[i]);
    }

    return ans;
}

// Driver code
int main()
{
    string s;
    s = ")))(((";

    cout << costToBalance(s) << endl;

    s = "))((";
    cout << costToBalance(s) << endl;

    return 0;
}
```

Java

```
 // Java code to calculate the
// minimum cost to make the
// given parentheses balanced
import java.io.*;

class GFG
{
    static int costToBalance(String s)
    {
        if (s.length() == 0)
            System.out.println(0);

        // To store absolute count
        // of balanced and unbalanced
        // parenthesis
        int ans = 0;

        // o(open bracket) stores count
        // of '(' and c(close bracket)
        // stores count of ')'
```

```
        int o = 0, c = 0;

        for (int i = 0; i < s.length(); i++)
        {
            if (s.charAt(i) == '(')
                o++;
            if (s.charAt(i) == ')')
                c++;
        }

        if (o != c)
            return -1;

        int []a = new int[s.length()];
        if (s.charAt(0) == '(')
            a[0] = 1;
        else
            a[0] = -1;

        if (a[0] < 0)
            ans += Math.abs(a[0]);

        for (int i = 1; i < s.length(); i++)
        {
            if (s.charAt(i) == '(')
                a[i] = a[i - 1] + 1;
            else
                a[i] = a[i - 1] - 1;
            if (a[i] < 0)
                ans += Math.abs(a[i]);
        }

        return ans;
    }

    // Driver code
    public static void main(String args[])
    {
        String s;
        s = ")))(((";

        System.out.println(costToBalance(s));

        s = "))((";
        System.out.println(costToBalance(s));
    }
}
```

```
// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
 // C# code to calculate the
// minimum cost to make the
// given parentheses balanced
using System;

class GFG
{
    static int costToBalance(string s)
    {
        if (s.Length == 0)
            Console.WriteLine(0);

        // To store absolute count
        // of balanced and unbalanced
        // parenthesis
        int ans = 0;

        // o(open bracket) stores count
        // of '(' and c(close bracket)
        // stores count of ')'
        int o = 0, c = 0;

        for (int i = 0; i < s.Length; i++)
        {
            if (s[i] == '(')
                o++;
            if (s[i] == ')')
                c++;
        }

        if (o != c)
            return -1;

        int []a = new int[s.Length];
        if (s[0] == '(')
            a[0] = 1;
        else
            a[0] = -1;

        if (a[0] < 0)
            ans += Math.Abs(a[0]);

        for (int i = 1; i < s.Length; i++)
```

```
        {
            if (s[i] == '(')
                a[i] = a[i - 1] + 1;
            else
                a[i] = a[i - 1] - 1;
            if (a[i] < 0)
                ans += Math.Abs(a[i]);
        }

        return ans;
    }

    // Driver code
    static void Main()
    {
        string s;
        s = ")))(((";

        Console.WriteLine (costToBalance(s));

        s = "))((";
        Console.WriteLine (costToBalance(s));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

```
9
4
```

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/cost-balance-parantheses/

Chapter 27

Count 1's in a sorted binary array

Count 1's in a sorted binary array - GeeksforGeeks

Given a binary array sorted in non-increasing order, count the number of 1's in it.

Examples:

```
Input: arr[] = {1, 1, 0, 0, 0, 0, 0}
Output: 2

Input: arr[] = {1, 1, 1, 1, 1, 1, 1}
Output: 7

Input: arr[] = {0, 0, 0, 0, 0, 0, 0}
Output: 0
```

A simple solution is to linearly traverse the array. The time complexity of the simple solution is O(n). We can use Binary Searchto find count in O(Logn) time. The idea is to look for last occurrence of 1 using Binary Search. Once we find the index last occurrence, we return index + 1 as count.

The following is the implementation of above idea.

C++

```
 // C++ program to count one's in a boolean array
#include <iostream>
using namespace std;

/* Returns counts of 1's in arr[low..high].  The array is
   assumed to be sorted in non-increasing order */
```

```
int countOnes(bool arr[], int low, int high)
{
  if (high >= low)
  {
    // get the middle index
    int mid = low + (high - low)/2;

    // check if the element at middle index is last 1
    if ( (mid == high || arr[mid+1] == 0) && (arr[mid] == 1))
      return mid+1;

    // If element is not last 1, recur for right side
    if (arr[mid] == 1)
      return countOnes(arr, (mid + 1), high);

    // else recur for left side
    return countOnes(arr, low, (mid -1));
  }
  return 0;
}

/* Driver program to test above functions */
int main()
{
   bool arr[] = {1, 1, 1, 1, 0, 0, 0};
   int n = sizeof(arr)/sizeof(arr[0]);
   cout << "Count of 1's in given array is " << countOnes(arr, 0, n-1);
   return 0;
}
```

Python

```
 # Python program to count one's in a boolean array

# Returns counts of 1's in arr[low..high].  The array is
# assumed to be sorted in non-increasing order
def countOnes(arr,low,high):

    if high>=low:

        # get the middle index
        mid = low + (high-low)/2

        # check if the element at middle index is last 1
        if ((mid == high or arr[mid+1]==0) and (arr[mid]==1)):
            return mid+1

        # If element is not last 1, recur for right side
```

```
        if arr[mid]==1:
            return countOnes(arr, (mid+1), high)

        # else recur for left side
        return countOnes(arr, low, mid-1)

    return 0

# Driver function
arr=[1, 1, 1, 1, 0, 0, 0]
print "Count of 1's in given array is",countOnes(arr, 0 , len(arr)-1)

# This code is contributed by __Devesh Agrawal__
```

Java

```
 // Java program to count 1's in a sorted array
class CountOnes
{
    /* Returns counts of 1's in arr[low..high].  The
       array is assumed to be sorted in non-increasing
       order */
    int countOnes(int arr[], int low, int high)
    {
      if (high >= low)
      {
        // get the middle index
        int mid = low + (high - low)/2;

        // check if the element at middle index is last 1
        if ( (mid == high || arr[mid+1] == 0) &&
             (arr[mid] == 1))
          return mid+1;

        // If element is not last 1, recur for right side
        if (arr[mid] == 1)
          return countOnes(arr, (mid + 1), high);

        // else recur for left side
        return countOnes(arr, low, (mid -1));
      }
      return 0;
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
       CountOnes ob = new CountOnes();
```

```
        int arr[] = {1, 1, 1, 1, 0, 0, 0};
        int n = arr.length;
        System.out.println("Count of 1's in given array is " +
                           ob.countOnes(arr, 0, n-1) );
    }
}
/* This code is contributed by Rajat Mishra */
```

C#

```
 // C# program to count 1's in a sorted array
using System;

class GFG {

    /* Returns counts of 1's in arr[low..high].
    The array is assumed to be sorted in
    non-increasing order */
    static int countOnes(int []arr, int low, int high)
    {
        if (high >= low)
        {

            // get the middle index
            int mid = low + (high - low) / 2;

            // check if the element at middle
            // index is last 1
            if ( (mid == high || arr[mid+1] == 0)
                              && (arr[mid] == 1))
            return mid+1;

            // If element is not last 1, recur
            // for right side
            if (arr[mid] == 1)
                return countOnes(arr, (mid + 1), high);

            // else recur for left side
            return countOnes(arr, low, (mid - 1));
        }

        return 0;
    }

    /* Driver program to test above functions */
    public static void Main()
    {
        int []arr = {1, 1, 1, 1, 0, 0, 0};
```

```
        int n = arr.Length;

        Console.WriteLine("Count of 1's in given "
          + "array is " + countOnes(arr, 0, n-1) );
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to count one's in a
// boolean array

/* Returns counts of 1's in arr[low..high].
The array is assumed to be sorted in
non-increasing order */
function countOnes( $arr, $low, $high)
{
    if ($high >= $low)
    {
        // get the middle index
        $mid = $low + ($high - $low)/2;

        // check if the element at middle
        // index is last 1
        if ( ($mid == $high or $arr[$mid+1] == 0)
                          and ($arr[$mid] == 1))
            return $mid+1;

        // If element is not last 1, recur for
        // right side
        if ($arr[$mid] == 1)
            return countOnes($arr, ($mid + 1),
                                        $high);

        // else recur for left side
        return countOnes($arr, $low, ($mid -1));
    }

    return 0;
}

/* Driver program to test above functions */
$arr = array(1, 1, 1, 1, 0, 0, 0);
$n = count($arr);
echo "Count of 1's in given array is " ,
```

```
                    countOnes($arr, 0, $n-1);

// This code is contributed by anuj_67.
?>
```

Output:

```
Count of 1's in given array is 4
```

Time complexity of the above solution is O(Logn)

Improved By : Sam007, vt_m

Source

https://www.geeksforgeeks.org/count-1s-sorted-binary-array/

Chapter 28

Count elements smaller than or equal to x in a sorted matrix

Count elements smaller than or equal to x in a sorted matrix - GeeksforGeeks

Given a n x n strictly sorted matrix and a value **x**. The problem is to count the elements smaller than or equal to **x** in the given matrix. Here strictly sorted matrix means that matrix is sorted in a way such that all elements in a row are sorted in increasing order and for row 'i', where 1 <= i <= n-1, first element of row 'i' is greater than or equal to the last element of row 'i-1'.

Examples:

```
Input : mat[][] = { {1, 4, 5},
                    {6, 10, 11},
                    {12, 15, 20} }
       x = 9
Output : 4
The elements smaller than '9' are:
1, 4, 5 and 6.

Input : mat[][] = { {1, 4, 7, 8},
                {10, 10, 12, 14},
            {15, 26, 30, 31},
            {31, 42, 46, 50} }
        x = 31
Output : 13
```

Naive Approach: Traverse the matrix using two nested loops and count the elements smaller than or equal to **x**. Time Complexity is of O(n^2).

Efficient Approach: As matrix is strictly sorted, use the concept of binary search technique. First apply the binary search technique on the elements of the first column to find

the row index number of the largest element smaller than equal to 'x'. For duplicates get the last row index no of occurrence of required element 'x'. Let it be **row_no.**. If no such row exists then return 0. Else apply the concept of binary search technique to find the column index no of the largest element smaller than or equal to 'x' in the row represented by **row_no**. Let it **col_no**. Finally return **((row_no) * n) + (col_no + 1).** The concept of binary search technique can be understood by the **binary_search** function of this post.

C++

```
 // C++ implementation to count elements smaller than
// or equal to x in a sorted matrix
#include <bits/stdc++.h>

using namespace std;

#define SIZE 100

// function returns the row index no of largest element
// smaller than equal to 'x' in first column of mat[][].
// For duplicates it returns the last row index no of
// occurrence of required element 'x'. If no such element
// exits then it returns -1.
int binarySearchOnRow(int mat[SIZE][SIZE],
                      int l, int h, int x)
{
    while (l <= h) {
        int mid = (l + h) / 2;

        // if 'x' is greater than or equal to mat[mid][0],
        // then search in mat[mid+1...h][0]
        if (mat[mid][0] <= x)
            l = mid + 1;

        // else search in mat[l...mid-1][0]
        else
            h = mid - 1;
    }

    // required row index number
    return h;
}

// function returns the column index no of largest element
// smaller than equal to 'x' in mat[row][].
// For duplicates it returns the last column index no of
// occurrence of required element 'x'.
int binarySearchOnCol(int mat[SIZE][SIZE],
                      int l, int h, int x, int row)
```

```
{
    while (l <= h) {
        int mid = (l + h) / 2;

        // if 'x' is greater than or equal to mat[row][mid],
        // then search in mat[row][mid+1...h]
        if (mat[row][mid] <= x)
            l = mid + 1;

        // else search in mat[row][l...mid-1]
        else
            h = mid - 1;
    }

    // required column index number
    return h;
}

// function to count elements smaller than
// or equal to x in a sorted matrix
int countSmallElements(int mat[SIZE][SIZE],
                       int n, int x)
{
    // to get the row index number of the largest element
    // smaller than equal to 'x' in mat[][]
    int row_no = binarySearchOnRow(mat, 0, n - 1, x);

    // if no such row exists
    if (row_no == -1)
        return 0;

    // to get the column index number of the largest element
    // smaller than equal to 'x' in mat[row_no][]
    int col_no = binarySearchOnCol(mat, 0, n - 1, x, row_no);

    // required count of elements
    return ((row_no)*n) + (col_no + 1);
}

// Driver program to test above
int main()
{
    int mat[SIZE][SIZE] = { { 1, 4, 7, 8 },
                            { 10, 10, 12, 14 },
                            { 15, 26, 30, 31 },
                            { 31, 42, 46, 50 } };
    int n = 4;
    int x = 31;
```

```
    cout << "Count = "
         << countSmallElements(mat, n, x);
    return 0;
}
```

Java

```
 //Java implementation to count elements
// smaller than or equal to x in a sorted matrix
import java.io.*;

class GFG {

static int SIZE=100;

// function returns the row index
//no of largest element smaller than
// equal to 'x' in first column of
//mat[][].For duplicates it returns
//the last row index no of
// occurrence of required element
// 'x'. If no such element
// exits then it returns -1.
static int binarySearchOnRow(int[][] mat,
                        int l, int h, int x)
{
    while (l <= h) {

        int mid = (l + h) / 2;

        // if 'x' is greater than or
        // equal to mat[mid][0],then
        // search in mat[mid+1...h][0]
        if (mat[mid][0] <= x)
            l = mid + 1;

        // else search in mat[l...mid-1][0]
        else
            h = mid - 1;
    }

    // required row index number
    return h;
}

// function returns the column index
// no of largest element
// smaller than equal to 'x' in
```

```
// mat[row][].For duplicates it returns
// the last column index no of
// occurrence of required element 'x'.
static int binarySearchOnCol(int[][] mat,
                    int l, int h, int x, int row)
{
    while (l <= h) {
        int mid = (l + h) / 2;

        // if 'x' is greater than or
        // equal to mat[row][mid], then
        // search in mat[row][mid+1...h]
        if (mat[row][mid] <= x)
            l = mid + 1;

        // else search in mat[row][l...mid-1]
        else
            h = mid - 1;
    }

    // required column index number
    return h;
}

// function to count elements smaller than
// or equal to x in a sorted matrix
static int countSmallElements(int[][] mat,
                                int n, int x)
{
    // to get the row index number of the largest
    // element smaller than equal to 'x' in mat[][]
    int row_no = binarySearchOnRow(mat,
                           0, n - 1, x);

    // if no such row exists
    if (row_no == -1)
        return 0;

    // to get the column index number of
    // the largest element smaller than
    // equal to 'x' in mat[row_no][]
    int col_no = binarySearchOnCol(mat,
                       0, n - 1, x, row_no);

    // required count of elements
    return ((row_no) * n) + (col_no + 1);
}
```

```
// Driver code
    public static void main (String[] args) {
        int mat[][] =      { { 1, 4, 7, 8 },
                             { 10, 10, 12, 14 },
                             { 15, 26, 30, 31 },
                             { 31, 42, 46, 50 } };
    int n = 4;
    int x = 31;
    System.out.println("Count = "
                + countSmallElements(mat, n, x));

    }
}
// This code is contributed by Gitanjali.
```

Python3

```
 # Python implementation to
# count elements smaller than
# or equal to x in a sorted matrix

SIZE = 100

# function returns the row index
# no of largest element
# smaller than equal to 'x' in
# first column of mat[][].
# For duplicates it returns the
# last row index no of
# occurrence of required element
# 'x'. If no such element
# exits then it returns -1.
def binarySearchOnRow(mat, l, h, x):
    while l <= h:
        mid = (l + h) // 2

        # if 'x' is greater than or
        # equal to mat[mid][0],
        # then search in mat[mid+1...h][0]
        if mat[mid][0] <= x:
            l = mid + 1

        # else search in mat[l...mid-1][0]
        else:
            h = mid - 1

    # required row index number
```

```
    return h

# function returns the column
# index no of largest element
# smaller than equal to 'x'
# in mat[row][].
# For duplicates it returns the
# last column index no of
# occurrence of required element 'x'.
def binarySearchOnCol(mat, l, h, x, row):
    while l <= h:
        mid = (l + h) // 2

        # if 'x' is greater than or
        # equal to mat[row][mid],
        # then search in mat[row][mid+1...h]
        if mat[row][mid] <= x:
            l = mid + 1

        # else search in mat[row][l...mid-1]
        else:
            h = mid - 1

    # required column index number
    return h

# function to count elements smaller than
# or equal to x in a sorted matrix
def countSmallElements(mat, n, x):
    # to get the row index number
    # of the largest element
    # smaller than equal to 'x' in mat[][]
    row_no = binarySearchOnRow(mat, 0, n - 1, x)

    # if no such row exists
    if row_no == -1:
        return 0

    # to get the column index number
    # of the largest element
    # smaller than equal to 'x' in mat[row_no][]
    col_no = binarySearchOnCol(mat, 0,
                               n - 1, x, row_no)

    # required count of elements
    return ((row_no)*n) + (col_no + 1)

# Driver program to test above
```

```
mat = [ [ 1, 4, 7, 8 ],
        [ 10, 10, 12, 14 ],
        [ 15, 26, 30, 31 ],
        [ 31, 42, 46, 50 ] ]
n = 4
x = 31
print("Count = " + str(countSmallElements(mat, n, x)))

# This code is contributed by Ansu Kumari.
```

C#

```
 //Java implementation to count elements
// smaller than or equal to x in a sorted matrix
using System;

class GFG {

    //static int SIZE=100;

    // function returns the row index
    //no of largest element smaller than
    // equal to 'x' in first column of
    //mat[][].For duplicates it returns
    //the last row index no of
    // occurrence of required element
    // 'x'. If no such element
    // exits then it returns -1.
    static int binarySearchOnRow(int [,] mat, int l,
                                 int h, int x)
    {
        while (l <= h) {

            int mid = (l + h) / 2;

            // if 'x' is greater than or
            // equal to mat[mid][0],then
            // search in mat[mid+1...h][0]
            if (mat[mid,0] <= x)
                l = mid + 1;

            // else search in mat[l...mid-1][0]
            else
                h = mid - 1;
        }

        // required row index number
        return h;
```

```
}

// function returns the column index
// no of largest element
// smaller than equal to 'x' in
// mat[row][].For duplicates it returns
// the last column index no of
// occurrence of required element 'x'.
static int binarySearchOnCol(int [,]mat, int l,
                                int h, int x, int row)
{
    while (l <= h) {
        int mid = (l + h) / 2;

        // if 'x' is greater than or
        // equal to mat[row][mid], then
        // search in mat[row][mid+1...h]
        if (mat[row,mid] <= x)
            l = mid + 1;

        // else search in mat[row][l...mid-1]
        else
            h = mid - 1;
    }

    // required column index number
    return h;
}

// function to count elements smaller than
// or equal to x in a sorted matrix
static int countSmallElements(int[,] mat,
                                int n, int x)
{
    // to get the row index number of the largest
    // element smaller than equal to 'x' in mat[][]
    int row_no = binarySearchOnRow(mat,
                            0, n - 1, x);

    // if no such row exists
    if (row_no == -1)
        return 0;

    // to get the column index number of
    // the largest element smaller than
    // equal to 'x' in mat[row_no][]
    int col_no = binarySearchOnCol(mat,
                0, n - 1, x, row_no);
```

```
        // required count of elements
        return ((row_no) * n) + (col_no + 1);
    }


    // Driver code
    public static void Main () {
        int [,]mat =     { { 1, 4, 7, 8 },
                           { 10, 10, 12, 14 },
                           { 15, 26, 30, 31 },
                           { 31, 42, 46, 50 } };
        int n = 4;
        int x = 31;
        Console.WriteLine("Count = "
                        + countSmallElements(mat, n, x));

    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP implementation to count
// elements smaller than or
// equal to x in a sorted matrix

// function returns the row index
// no of largest element smaller
// than equal to 'x' in first
// column of mat[][]. For duplicates
// it returns the last row index no.
// of occurrence of required element
// 'x'. If no such element exits then
// it returns -1.
function binarySearchOnRow($mat, $l,
                           $h, $x)
{
    while ($l <= $h) {
        $mid = floor(($l + $h) / 2);

        // if 'x' is greater than
        // or equal to mat[mid][0],
        // then search in
        // mat[mid+1...h][0]
        if ($mat[$mid][0] <= $x)
```

```
            $l = $mid + 1;

        // else search in
        // mat[l...mid-1][0]
        else
            $h = $mid - 1;
    }

    // required row index number
    return $h;
}

// function returns the column
// index no of largest element
// smaller than equal to 'x'
// in mat[row][]. For duplicates
// it returns the last column
// index no of occurrence of
// required element 'x'.
function binarySearchOnCol($mat, $l,
                        $h, $x, $row)
{
    while ($l <= $h) {
        $mid = floor(($l + $h) / 2);

        // if 'x' is greater than or
        // equal to mat[row][mid],
        // then search in
        // mat[row][mid+1...h]
        if ($mat[$row][$mid] <= $x)
            $l = $mid + 1;

        // else search in
        // mat[row][l...mid-1]
        else
            $h = $mid - 1;
    }

    // required column
    // index number
    return $h;
}

// function to count elements
// smaller than or equal to x
// in a sorted matrix
function countSmallElements($mat,
                            $n, $x)
```

```
{

    // to get the row index number
    // of the largest element
    // smaller than equal to 'x'
    // in mat[][]
    $row_no = binarySearchOnRow($mat, 0,
                            $n - 1, $x);

    // if no such row exists
    if ($row_no == -1)
        return 0;

    // to get the column index
    // number of the largest element
    // smaller than equal to 'x'
    // in mat[row_no][]
    $col_no = binarySearchOnCol($mat, 0,
                    $n - 1, $x, $row_no);

    // required count of elements
    return floor(($row_no) * $n) +
                     ($col_no + 1);
}

    // Driver Code
    $mat = array(array(1, 4, 7, 8),
                 array(10, 10, 12, 14),
                 array(15, 26, 30, 31),
                 array(31, 42, 46, 50));
    $n = 4;
    $x = 31;
    echo "Count = ", countSmallElements ($mat, $n, $x);

// This code is contributed by nitin mittal.
?>
```

Output:

```
Count = 13
```

Time Complexity: O(Log_2n).

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/count-elements-smaller-equal-x-sorted-matrix/

Chapter 29

Count items common to both the lists but with different prices

Count items common to both the lists but with different prices - GeeksforGeeks

Given two lists **list1** and **list2** containing **m** and **n** items respectively. Each item is associated with two fields: name and price. The problem is to count items which are in both the lists but with different prices.

Examples:

```
Input : list1[] = {{"apple", 60}, {"bread", 20},
                   {"wheat", 50}, {"oil", 30}}
    list2[] = {{"milk", 20}, {"bread", 15},
                   {"wheat", 40}, {"apple", 60}}
Output : 2
bread and wheat are the two items common to both the
lists but with different prices.
```

Source: Cognizant Interview Experience | Set 5.

Method 1 (Naive Approach): Using two nested loops compare each item of **list1** with all the items of **list2**. If match is found with a different price then increment the **count**.

CPP

```
 // C++ implementation to count items common to both
// the lists but with different prices
#include <bits/stdc++.h>
```

```
using namespace std;

// details of an item
struct item
{
    string name;
    int price;
};

// function to count items common to both
// the lists but with different prices
int countItems(item list1[], int m,
               item list2[], int n)
{
    int count = 0;

    // for each item of 'list1' check if it is in 'list2'
    // but with a different price
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)

            if ((list1[i].name.compare(list2[j].name) == 0) &&
                (list1[i].price != list2[j].price))
                count++;

    // required count of items
    return count;
}

// Driver program to test above
int main()
{
    item list1[] = {{"apple", 60}, {"bread", 20},
                    {"wheat", 50}, {"oil", 30}};
    item list2[] = {{"milk", 20}, {"bread", 15},
                   {"wheat", 40}, {"apple", 60}};

    int m = sizeof(list1) / sizeof(list1[0]);
    int n = sizeof(list2) / sizeof(list2[0]);

    cout << "Count = "
         << countItems(list1, m, list2, n);

    return 0;
}
```

Python3

```
 # Python implementation to
# count items common to both
# the lists but with different
# prices

# function to count items
# common to both
# the lists but with different prices
def countItems(list1, list2):
    count = 0

    # for each item of 'list1'
    # check if it is in 'list2'
    # but with a different price
    for i in list1:
        for j in list2:

            if i[0] == j[0] and i[1] != j[1]:
                count += 1

    # required count of items
    return count

# Driver program to test above
list1 = [("apple", 60), ("bread", 20),
            ("wheat", 50), ("oil", 30)]
list2 = [("milk", 20), ("bread", 15),
            ("wheat", 40), ("apple", 60)]

print("Count = ", countItems(list1, list2))

# This code is contributed by Ansu Kumari.
```

Output:

```
Count = 2
```

Time Complexity: O(m*n).
Auxiliary Space: O(1).

Method 2 (Binary Search): Sort the **list2** in alphabetical order of its items name. Now, for each item of **list1** check whether it in present in **list2** using binary search technique and get its price from **list2**. If prices are different the increment the **count**.

```
 // C++ implementation to count items common to both
// the lists but with different prices
#include <bits/stdc++.h>
```

```
using namespace std;

// details of an item
struct item
{
    string name;
    int price;
};

// comparator function used for sorting
bool compare(struct item a, struct item b)
{
    return (a.name.compare(b.name) <= 0);
}

// function to search 'str' in 'list2[]'. If it exists then
// price associated with 'str' in 'list2[]' is being
// returned else -1 is returned. Here binary serach
// trechnique is being applied for searching
int binary_search(item list2[], int low, int high, string str)
{
    while (low <= high)
    {
        int mid = (low + high) / 2;

        // if true the item 'str' is in 'list2'
        if (list2[mid].name.compare(str) == 0)
            return list2[mid].price;

        else if (list2[mid].name.compare(str) < 0)
            low = mid + 1;

        else
            high = mid - 1;
    }

    // item 'str' is not in 'list2'
    return -1;
}

// function to count items common to both
// the lists but with different prices
int countItems(item list1[], int m,
               item list2[], int n)
{
    // sort 'list2' in alphabetcal order of
    // items name
```

```
    sort(list2, list2 + n, compare);

    // initial count
    int count = 0;

    for (int i = 0; i < m; i++) {

        // get the price of item 'list1[i]' from 'list2'
        // if item in not present in second list then
        // -1 is being obtained
        int r = binary_search(list2, 0, n-1, list1[i].name);

        // if item is present in list2 with a
        // different price
        if ((r != -1) && (r != list1[i].price))
            count++;
    }

    // required count of items
    return count;
}

// Driver program to test above
int main()
{
    item list1[] = {{"apple", 60}, {"bread", 20},
                    {"wheat", 50}, {"oil", 30}};
    item list2[] = {{"milk", 20}, {"bread", 15},
                   {"wheat", 40}, {"apple", 60}};

    int m = sizeof(list1) / sizeof(list1[0]);
    int n = sizeof(list2) / sizeof(list2[0]);

    cout << "Count = "
         << countItems(list1, m, list2, n);

    return 0;
}
```

Output:

```
Count = 2
```

Time Complexity: ($m*\log_2 n$).
Auxiliary Space: O(1).
For efficiency, the list with maximum number of elements should sorted and used for binary search.

Method 3 (Efficient Approach): Create a hash table with **(key, value)** tuple as **(item name, price)**. Insert the elements of **list1** in the hash table. Now, for each element of **list2** check if it is the hash table or not. If it is present, then check if its price is different then the value from the hash table. If so then increment the **count**.

```
 // C++ implementation to count items common to both
// the lists but with different prices
#include <bits/stdc++.h>

using namespace std;

// details of an item
struct item
{
    string name;
    int price;
};

// function to count items common to both
// the lists but with different prices
int countItems(item list1[], int m,
               item list2[], int n)
{
    // 'um' implemented as hash table that contains
    // item name as the key and price as the value
    // associated with the key
    unordered_map<string, int> um;
    int count = 0;

    // insert elements of 'list1' in 'um'
    for (int i = 0; i < m; i++)
        um[list1[i].name] = list1[i].price;

    // for each element of 'list2' check if it is
    // present in 'um' with a different price
    // value
    for (int i = 0; i < n; i++)
        if ((um.find(list2[i].name) != um.end()) &&
            (um[list2[i].name] != list2[i].price))
            count++;

    // required count of items
    return count;
}

// Driver program to test above
int main()
{
```

```
    item list1[] = {{"apple", 60}, {"bread", 20},
                    {"wheat", 50}, {"oil", 30}};
    item list2[] = {{"milk", 20}, {"bread", 15},
                    {"wheat", 40}, {"apple", 60}};

    int m = sizeof(list1) / sizeof(list1[0]);
    int n = sizeof(list2) / sizeof(list2[0]);

    cout << "Count = "
         << countItems(list1, m, list2, n);

    return 0;
}
```

Output:

```
Count = 2
```

Time Complexity: O(m + n).
Auxiliary Space: O(m).
For efficiency, the list having minimum number of elements should be inserted in the hash table.

Source

https://www.geeksforgeeks.org/count-items-common-lists-different-prices/

Chapter 30

Count number of elements between two given elements in array

Count number of elements between two given elements in array - GeeksforGeeks

Given an unsorted array of n elements and also given two points num1 and num2. The task is to count number of elements occurs between the given points (excluding num1 and num2).

If there are multiple occurrences of num1 and num2, we need to consider leftmost occurrence of num1 and rightmost occurrence of num2.

Examples:

```
Input : arr[] = {3 5 7 6 4 9 12 4 8}
        num1 = 5
        num2 = 4
Output : 5
Number of elements between leftmost occurrence
of 5 and rightmost occurrence of 4 is five.

Input : arr[] = {4, 6, 8, 3, 6, 2, 8, 9, 4}
        num1 = 4
        num2 = 4
Output : 7

Input : arr[] = {4, 6, 8, 3, 6, 2, 8, 9, 4}
        num1 = 4
        num2 = 10
Output : 0
```

The solution should traverse array only once in all cases (when single or both elements are not present)

The idea is to traverse array from left and find first occurrence of num1. If we reach end, we return 0. Then we traverse from rightmost element and find num2. We traverse only till the point which is greater than index of num1. If we reach end, we return 0. If we found both elements, we return count using indexes of found elements.

CPP

```
 // Program to count number of elements between
// two given elements.
#include <bits/stdc++.h>
using namespace std;

// Function to count number of elements
// occurs between the elements.
int getCount(int arr[], int n, int num1, int num2)
{
    // Find num1
    int i = 0;
    for (i = 0; i < n; i++)
        if (arr[i] == num1)
            break;

    // If num1 is not present or present at end
    if (i >= n-1)
        return 0;

    // Find num2
    int j;
    for (j = n-1; j >= i+1; j--)
        if (arr[j] == num2)
            break;

    // If num2 is not present
    if (j == i)
        return 0;

    // return number of elements between
    // the two elements.
    return (j - i - 1);
}

// Driver Code
int main()
{
    int arr[] = { 3, 5, 7, 6, 4, 9, 12, 4, 8 };
```

```
    int n = sizeof(arr) / sizeof(arr[0]);
    int num1 = 5, num2 = 4;
    cout << getCount(arr, n, num1, num2);
    return 0;
}
```

Java

```
 // Program to count number of elements
// between two given elements.
import java.io.*;

class GFG
{
    // Function to count number of elements
    // occurs between the elements.
    static int getCount(int arr[], int n,
                            int num1, int num2)
    {
        // Find num1
        int i = 0;
        for (i = 0; i < n; i++)
            if (arr[i] == num1)
                break;

        // If num1 is not present
        // or present at end
        if (i >= n - 1)
            return 0;

        // Find num2
        int j;
        for (j = n - 1; j >= i + 1; j--)
            if (arr[j] == num2)
                break;

        // If num2 is not present
        if (j == i)
            return 0;

        // return number of elements
        // between the two elements.
        return (j - i - 1);
    }

    // Drvier program
    public static void main (String[] args)
    {
```

```
        int arr[] = { 3, 5, 7, 6, 4, 9, 12, 4, 8 };
        int n = arr.length;
        int num1 = 5, num2 = 4;
        System.out.println( getCount(arr, n, num1, num2));

    }
}
// This code is contributed by vt_m
```

Python3

```
 # Python Program to count number of elements between
# two given elements.

# Function to count number of elements
# occurs between the elements.
def getCount(arr, n, num1, num2):

    # Find num1
    for i in range(0,n):
        if (arr[i] == num1):
            break

    #If num1 is not present or present at end
    if (i >= n-1):
        return 0

    # Find num2
    for j in range(n-1, i+1, -1):
        if (arr[j] == num2):
            break

    # If num2 is not present
    if (j == i):
        return 0

    # return number of elements between
    # the two elements.
    return (j - i - 1)

# Driver Code
arr= [ 3, 5, 7, 6, 4, 9, 12, 4, 8 ]
n=len(arr)
num1 = 5
num2 = 4
print(getCount(arr, n, num1, num2))

# This code is contributed by SHARIQ_JMI
```

C#

```
 // C# Program to count number of elements
// between two given elements.
using System;

class GFG  {

    // Function to count number of elements
    // occurs between the elements.
    static int getCount(int []arr, int n,
                        int num1, int num2)
    {

        // Find num1
        int i = 0;
        for (i = 0; i < n; i++)
            if (arr[i] == num1)
                break;

        // If num1 is not present
        // or present at end
        if (i >= n - 1)
            return 0;

        // Find num2
        int j;
        for (j = n - 1; j >= i + 1; j--)
            if (arr[j] == num2)
                break;

        // If num2 is not present
        if (j == i)
            return 0;

        // return number of elements
        // between the two elements.
        return (j - i - 1);
    }

    // Driver Code
    public static void Main ()
    {
        int []arr = {3, 5, 7, 6, 4, 9, 12, 4, 8};
        int n = arr.Length;
        int num1 = 5, num2 = 4;
        Console.WriteLine(getCount(arr, n, num1, num2));
```

```
    }
}

// This article is contributed by vt_m.
```

PHP

```
 <?php
// Program to count number
// of elements between
// two given elements.

// Function to count
// number of elements
// occurs between the
// elements.
function getCount($arr, $n,
                  $num1, $num2)
{

    // Find num1
    $i = 0;
    for ($i = 0; $i < $n; $i++)
        if ($arr[$i] == $num1)
            break;

    // If num1 is not present
    // or present at end
    if ($i >= $n - 1)
        return 0;

    // Find num2
    $j;
    for ($j = $n - 1; $j >= $i + 1; $j--)
        if ($arr[$j] == $num2)
            break;

    // If num2 is not present
    if ($j == $i)
        return 0;

    // return number of elements
    // betweenthe two elements.
    return ($j - $i - 1);
}

// Driver Code
$arr = array(3, 5, 7, 6, 4, 9, 12, 4, 8);
```

```
$n = sizeof($arr);
$num1 = 5; $num2 = 4;
echo(getCount($arr, $n, $num1, $num2));

// This code is contributed by Ajit.
?>
```

Output:

```
5
```

How to handle multiple queries?
For handling multiple queries, we can use hashing and store leftmost and rightmost indexes for every element present in array. Once we have stored these, we can answer all queries in O(1) time.

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/count-number-elements-two-given-elements-array/

Chapter 31

Count number of occurrences (or frequency) in a sorted array

Count number of occurrences (or frequency) in a sorted array - GeeksforGeeks

Given a sorted array arr[] and a number x, write a function that counts the occurrences of x in arr[]. Expected time complexity is O(Logn)

Examples:

```
Input: arr[] = {1, 1, 2, 2, 2, 2, 3,},   x = 2
Output: 4 // x (or 2) occurs 4 times in arr[]

Input: arr[] = {1, 1, 2, 2, 2, 2, 3,},   x = 3
Output: 1

Input: arr[] = {1, 1, 2, 2, 2, 2, 3,},   x = 1
Output: 2

Input: arr[] = {1, 1, 2, 2, 2, 2, 3,},   x = 4
Output: -1 // 4 doesn't occur in arr[]
```

Method 1 (Linear Search)
Linearly search for x, count the occurrences of x and return the count.

C++

```
 // C++ program to count occurrences of an element
#include<bits/stdc++.h>
using namespace std;

// Returns number of times x occurs in arr[0..n-1]
```

```
int countOccurrences(int arr[], int n, int x)
{
    int res = 0;
    for (int i=0; i<n; i++)
        if (x == arr[i])
          res++;
    return res;
}

// Driver code
int main()
{
    int arr[] = {1, 2, 2, 2, 2, 3, 4, 7 ,8 ,8 };
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 2;
    cout << countOccurrences(arr, n, x);
    return 0;
}
```

Java

```
 // Java program to count occurrences
// of an element

class Main
{
    // Returns number of times x occurs in arr[0..n-1]
    static int countOccurrences(int arr[], int n, int x)
    {
        int res = 0;
        for (int i=0; i<n; i++)
            if (x == arr[i])
              res++;
        return res;
    }

    public static void main(String args[])
    {
        int arr[] = {1, 2, 2, 2, 2, 3, 4, 7 ,8 ,8 };
        int n = arr.length;
        int x = 2;
        System.out.println(countOccurrences(arr, n, x));
    }
}
```

Python3

```
 # Python3 program to count
```

```
# occurrences of an element

# Returns number of times x
# occurs in arr[0..n-1]
def countOccurrences(arr, n, x):
    res = 0
    for i in range(n):
        if x == arr[i]:
            res += 1
    return res

# Driver code
arr = [1, 2, 2, 2, 2, 3, 4, 7 ,8 ,8]
n = len(arr)
x = 2
print (countOccurrences(arr, n, x))
```

C#

```
 // C# program to count occurrences
// of an element
using System;

class GFG
{
    // Returns number of times x
    // occurs in arr[0..n-1]
    static int countOccurrences(int []arr,
                                int n, int x)
    {
        int res = 0;

        for (int i = 0; i < n; i++)
            if (x == arr[i])
            res++;

        return res;
    }

    // driver code
    public static void Main()
    {
        int []arr = {1, 2, 2, 2, 2, 3, 4, 7 ,8 ,8 };
        int n = arr.Length;
        int x = 2;

        Console.Write(countOccurrences(arr, n, x));
    }
```

```
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to count occurrences
// of an element

// Returns number of times x
// occurs in arr[0..n-1]
function countOccurrences($arr, $n, $x)
{
    $res = 0;
    for ($i = 0; $i < $n; $i++)
        if ($x == $arr[$i])
        $res++;
    return $res;
}

    // Driver code
    $arr = array(1, 2, 2, 2, 2, 3, 4, 7 ,8 ,8 );
    $n = count($arr);
    $x = 2;
    echo countOccurrences($arr,$n, $x);

// This code is contributed by Sam007
?>
```

Output :

```
4
```

Time Complexity: O(n)

Method 2 (Better using Binary Search)
We first find an occurrence using binary search. Then we match toward left and right sides of the matched the found index.

C++

```
 // C++ program to count occurrences of an element
#include <bits/stdc++.h>
using namespace std;
```

```
// A recursive binary search function. It returns
// location of x in given array arr[l..r] is present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    if (r < l)
        return -1;

    int mid = l + (r - l) / 2;

    // If the element is present at the middle
    // itself
    if (arr[mid] == x)
        return mid;

    // If element is smaller than mid, then
    // it can only be present in left subarray
    if (arr[mid] > x)
        return binarySearch(arr, l, mid - 1, x);

    // Else the element can only be present
    // in right subarray
    return binarySearch(arr, mid + 1, r, x);
}

// Returns number of times x occurs in arr[0..n-1]
int countOccurrences(int arr[], int n, int x)
{
    int ind = binarySearch(arr, 0, n - 1, x);

    // If element is not present
    if (ind == -1)
        return 0;

    // Count elements on left side.
    int count = 1;
    int left = ind - 1;
    while (left >= 0 && arr[left] == x)
        count++, left--;

    // Count elements on right side.
    int right = ind + 1;
    while (right < n && arr[right] == x)
        count++, right++;

    return count;
}
```

```
// Driver code
int main()
{
    int arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 2;
    cout << countOccurrences(arr, n, x);
    return 0;
}
```

Java

```
 // Java program to count
// occurrences of an element
class GFG
{

    // A recursive binary search
    // function. It returns location
    // of x in given array arr[l..r]
    // is present, otherwise -1
    static int binarySearch(int arr[], int l,
                            int r, int x)
    {
        if (r < l)
            return -1;

        int mid = l + (r - l) / 2;

        // If the element is present
        // at the middle itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than
        // mid, then it can only be
        // present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l,
                                mid - 1, x);

        // Else the element can
        // only be present in
        // right subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // Returns number of times x
```

```
    // occurs in arr[0..n-1]
    static int countOccurrences(int arr[],
                                int n, int x)
    {
        int ind = binarySearch(arr, 0,
                               n - 1, x);

        // If element is not present
        if (ind == -1)
            return 0;

        // Count elements on left side.
        int count = 1;
        int left = ind - 1;
        while (left >= 0 &&
               arr[left] == x)
        {
            count++;
            left--;
        }

        // Count elements
        // on right side.
        int right = ind + 1;
        while (right < n &&
               arr[right] == x)
        {
            count++;
            right++;
        }

        return count;
    }


    // Driver code
    public static void main(String[] args)
    {
        int arr[] = {1, 2, 2, 2, 2,
                     3, 4, 7, 8, 8};
        int n = arr.length;
        int x = 2;
        System.out.print(countOccurrences(arr, n, x));
    }
}

// This code is contributed
// by ChitraNayal
```

Python 3

```
 # Python 3 program to count
# occurrences of an element

# A recursive binary search
# function. It returns location
# of x in given array arr[l..r]
# is present, otherwise -1
def binarySearch(arr, l, r, x):
    if (r < l):
        return -1

    mid = int( l + (r - l) / 2)

    # If the element is present
    # at the middle itself
    if arr[mid] == x:
        return mid

    # If element is smaller than
    # mid, then it can only be
    # present in left subarray
    if arr[mid] > x:
        return binarySearch(arr, l,
                            mid - 1, x)

    # Else the element
    # can only be present
    # in right subarray
    return binarySearch(arr, mid + 1,
                                r, x)

# Returns number of times
# x occurs in arr[0..n-1]
def countOccurrences(arr, n, x):
    ind = binarySearch(arr, 0, n - 1, x)

    # If element is not present
    if ind == -1:
        return 0

    # Count elements
    # on left side.
    count = 1
    left = ind - 1
    while (left >= 0 and
           arr[left] == x):
```

```
        count += 1
        left -= 1

    # Count elements on
    # right side.
    right = ind + 1;
    while (right < n and
           arr[right] == x):
        count += 1
        right += 1

    return count

# Driver code
arr = [ 1, 2, 2, 2, 2,
        3, 4, 7, 8, 8 ]
n = len(arr)
x = 2
print(countOccurrences(arr, n, x))

# This code is contributed
# by ChitraNayal
```

C#

```
 // C# program to count
// occurrences of an element
using System;

class GFG
{

    // A recursive binary search
    // function. It returns location
    // of x in given array arr[l..r]
    // is present, otherwise -1
    static int binarySearch(int[] arr, int l,
                            int r, int x)
    {
        if (r < l)
            return -1;

        int mid = l + (r - l) / 2;

        // If the element is present
        // at the middle itself
        if (arr[mid] == x)
            return mid;
```

```
    // If element is smaller than
    // mid, then it can only be
    // present in left subarray
    if (arr[mid] > x)
        return binarySearch(arr, l,
                            mid - 1, x);

    // Else the element
    // can only be present
    // in right subarray
    return binarySearch(arr, mid + 1,
                                r, x);
}

// Returns number of times x
// occurs in arr[0..n-1]
static int countOccurrences(int[] arr,
                            int n, int x)
{
    int ind = binarySearch(arr, 0,
                           n - 1, x);

    // If element is not present
    if (ind == -1)
        return 0;

    // Count elements on left side.
    int count = 1;
    int left = ind - 1;
    while (left >= 0 &&
           arr[left] == x)
    {
        count++;
        left--;
    }

    // Count elements on right side.
    int right = ind + 1;
    while (right < n &&
           arr[right] == x)
    {
        count++;
        right++;
    }

    return count;
}
```

```
    // Driver code
    public static void Main()
    {
        int[] arr = {1, 2, 2, 2, 2,
                     3, 4, 7, 8, 8};
        int n = arr.Length;
        int x = 2;
        Console.Write(countOccurrences(arr, n, x));
    }
}

// This code is contributed
// by ChitraNayal
```

PHP

```
 <?php
// PHP program to count
// occurrences of an element

// A recursive binary search
// function. It returns location
// of x in given array arr[l..r]
// is present, otherwise -1
function binarySearch(&$arr, $l,
                      $r, $x)
{
    if ($r < $l)
        return -1;

    $mid = $l + ($r - $l) / 2;

    // If the element is present
    // at the middle itself
    if ($arr[$mid] == $x)
        return $mid;

    // If element is smaller than
    // mid, then it can only be
    // present in left subarray
    if ($arr[$mid] > $x)
        return binarySearch($arr, $l,
                            $mid - 1, $x);

    // Else the element
    // can only be present
```

```
    // in right subarray
    return binarySearch($arr, $mid + 1,
                                $r, $x);
}

// Returns number of times
// x occurs in arr[0..n-1]
function countOccurrences($arr, $n, $x)
{
    $ind = binarySearch($arr, 0,
                        $n - 1, $x);

    // If element is not present
    if ($ind == -1)
        return 0;

    // Count elements
    // on left side.
    $count = 1;
    $left = $ind - 1;
    while ($left >= 0 &&
           $arr[$left] == $x)
    {
        $count++;
        $left--;
    }

    // Count elements on right side.
    $right = $ind + 1;
    while ($right < $n &&
           $arr[$right] == $x)
    {
        $count++;
        $right++;
    }
    return $count;
}

// Driver code
$arr = array( 1, 2, 2, 2, 2,
              3, 4, 7, 8, 8 );
$n = sizeof($arr);
$x = 2;
echo countOccurrences($arr, $n, $x);

// This code is contributed
// by ChitraNayal
?>
```

Output :

```
4
```

Time Complexity : O(Log n + count) where count is number of occurrences.

Method 3 (Best using Improved Binary Search)
1) Use Binary search to get index of the first occurrence of x in arr[]. Let the index of the first occurrence be i.
2) Use Binary search to get index of the last occurrence of x in arr[]. Let the index of the last occurrence be j.
3) Return (j – i + 1);

C++

```
 // C++ program to count occurrences of an element
// in a sorted array.
# include <bits/stdc++.h>
using namespace std;

/* if x is present in arr[] then returns the count
    of occurrences of x, otherwise returns 0. */
int count(int arr[], int x, int n)
{
  /* get the index of first occurrence of x */
  int *low = lower_bound(arr, arr+n, x);

  // If element is not present, return 0
  if (low == (arr + n) || *low != x)
     return 0;

  /* Else get the index of last occurrence of x.
     Note that we  are only looking in the
     subarray after first occurrence */
  int *high = upper_bound(low, arr+n, x);

  /* return count */
  return high - low;
}

/* driver program to test above functions */
int main()
{
  int arr[] = {1, 2, 2, 3, 3, 3, 3};
  int x =  3;  // Element to be counted in arr[]
  int n = sizeof(arr)/sizeof(arr[0]);
```

```
  int c = count(arr, x, n);
  printf(" %d occurs %d times ", x, c);
  return 0;
}
```

C

```
 # include <stdio.h>

/* if x is present in arr[] then returns
   the index of FIRST occurrence
   of x in arr[0..n-1], otherwise returns -1 */
int first(int arr[], int low, int high, int x, int n)
{
  if(high >= low)
  {
    int mid = (low + high)/2;  /*low + (high - low)/2;*/
    if( ( mid == 0 || x > arr[mid-1]) && arr[mid] == x)
      return mid;
    else if(x > arr[mid])
      return first(arr, (mid + 1), high, x, n);
    else
      return first(arr, low, (mid -1), x, n);
  }
  return -1;
}

/* if x is present in arr[] then returns the
   index of LAST occurrence of x in arr[0..n-1],
   otherwise returns -1 */
int last(int arr[], int low, int high, int x, int n)
{
  if (high >= low)
  {
    int mid = (low + high)/2;  /*low + (high - low)/2;*/
    if( ( mid == n-1 || x < arr[mid+1]) && arr[mid] == x )
      return mid;
    else if(x < arr[mid])
      return last(arr, low, (mid -1), x, n);
    else
      return last(arr, (mid + 1), high, x, n);
  }
  return -1;
}

/* if x is present in arr[] then returns the count
   of occurrences of x, otherwise returns -1. */
int count(int arr[], int x, int n)
```

```
{
  int i; // index of first occurrence of x in arr[0..n-1]
  int j; // index of last occurrence of x in arr[0..n-1]

  /* get the index of first occurrence of x */
  i = first(arr, 0, n-1, x, n);

  /* If x doesn't exist in arr[] then return -1 */
  if(i == -1)
    return i;

  /* Else get the index of last occurrence of x.
     Note that we are only looking in the subarray
     after first occurrence */
  j = last(arr, i, n-1, x, n);

  /* return count */
  return j-i+1;
}

/* driver program to test above functions */
int main()
{
  int arr[] = {1, 2, 2, 3, 3, 3, 3};
  int x =  3;  // Element to be counted in arr[]
  int n = sizeof(arr)/sizeof(arr[0]);
  int c = count(arr, x, n);
  printf(" %d occurs %d times ", x, c);
  getchar();
  return 0;
}
```

Java

```
 // Java program to count occurrences
// of an element

class Main
{
    /* if x is present in arr[] then returns
       the count of occurrences of x,
       otherwise returns -1. */
    static int count(int arr[], int x, int n)
    {
      // index of first occurrence of x in arr[0..n-1]
      int i;

      // index of last occurrence of x in arr[0..n-1]
```

```
  int j;

  /* get the index of first occurrence of x */
  i = first(arr, 0, n-1, x, n);

  /* If x doesn't exist in arr[] then return -1 */
  if(i == -1)
    return i;

  /* Else get the index of last occurrence of x.
     Note that we are only looking in the
     subarray after first occurrence */
  j = last(arr, i, n-1, x, n);

  /* return count */
  return j-i+1;
}

/* if x is present in arr[] then returns the
   index of FIRST occurrence of x in arr[0..n-1],
   otherwise returns -1 */
static int first(int arr[], int low, int high, int x, int n)
{
  if(high >= low)
  {
    /*low + (high - low)/2;*/
    int mid = (low + high)/2;
    if( ( mid == 0 || x > arr[mid-1]) && arr[mid] == x)
      return mid;
    else if(x > arr[mid])
      return first(arr, (mid + 1), high, x, n);
    else
      return first(arr, low, (mid -1), x, n);
  }
  return -1;
}

/* if x is present in arr[] then returns the
   index of LAST occurrence of x in arr[0..n-1],
   otherwise returns -1 */
static int last(int arr[], int low, int high, int x, int n)
{
  if(high >= low)
  {
    /*low + (high - low)/2;*/
    int mid = (low + high)/2;
    if( ( mid == n-1 || x < arr[mid+1]) && arr[mid] == x )
      return mid;
```

```
        else if(x < arr[mid])
          return last(arr, low, (mid -1), x, n);
        else
          return last(arr, (mid + 1), high, x, n);
      }
      return -1;
    }

    public static void main(String args[])
    {
        int arr[] = {1, 2, 2, 3, 3, 3, 3};

        // Element to be counted in arr[]
        int x =  3;
        int n = arr.length;
        int c = count(arr, x, n);
        System.out.println(x+" occurs "+c+" times");
    }
}
```

Python3

```
 # Python3 program to count
# occurrences of an element

# if x is present in arr[] then
# returns the count of occurrences
# of x, otherwise returns -1.
def count(arr, x, n):

    # get the index of first
    # occurrence of x
    i = first(arr, 0, n-1, x, n)

    # If x doesn't exist in
    # arr[] then return -1
    if i == -1:
        return i

    # Else get the index of last occurrence
    # of x. Note that we are only looking
    # in the subarray after first occurrence
    j = last(arr, i, n-1, x, n);

    # return count
    return j-i+1;

# if x is present in arr[] then return
```

```
# the index of FIRST occurrence of x in
# arr[0..n-1], otherwise returns -1
def first(arr, low, high, x, n):
    if high >= low:

        # low + (high - low)/2
        mid = (low + high)//2

    if (mid == 0 or x > arr[mid-1]) and arr[mid] == x:
        return mid
    elif x > arr[mid]:
        return first(arr, (mid + 1), high, x, n)
    else:
        return first(arr, low, (mid -1), x, n)
    return -1;

# if x is present in arr[] then return
# the index of LAST occurrence of x
# in arr[0..n-1], otherwise returns -1
def last(arr, low, high, x, n):
    if high >= low:

        # low + (high - low)/2
        mid = (low + high)//2;

    if(mid == n-1 or x < arr[mid+1]) and arr[mid] == x :
        return mid
    elif x < arr[mid]:
        return last(arr, low, (mid -1), x, n)
    else:
        return last(arr, (mid + 1), high, x, n)
    return -1

# driver program to test above functions
arr = [1, 2, 2, 3, 3, 3, 3]
x = 3  # Element to be counted in arr[]
n = len(arr)
c = count(arr, x, n)
print ("%d occurs %d times "%(x, c))
```

C#

```
 // C# program to count occurrences
// of an element
using System;

class GFG
{
```

```
/* if x is present in arr[] then returns
the count of occurrences of x,
otherwise returns -1. */
static int count(int []arr, int x, int n)
{
// index of first occurrence of x in arr[0..n-1]
int i;

// index of last occurrence of x in arr[0..n-1]
int j;

/* get the index of first occurrence of x */
i = first(arr, 0, n-1, x, n);

/* If x doesn't exist in arr[] then return -1 */
if(i == -1)
    return i;

/* Else get the index of last occurrence of x.
    Note that we are only looking in the
    subarray after first occurrence */
j = last(arr, i, n-1, x, n);

/* return count */
return j-i+1;
}

/* if x is present in arr[] then returns the
index of FIRST occurrence of x in arr[0..n-1],
otherwise returns -1 */
static int first(int []arr, int low, int high,
                                  int x, int n)
{
if(high >= low)
{
    /*low + (high - low)/2;*/
    int mid = (low + high)/2;
    if( ( mid == 0 || x > arr[mid-1])
                        && arr[mid] == x)
    return mid;
    else if(x > arr[mid])
    return first(arr, (mid + 1), high, x, n);
    else
    return first(arr, low, (mid -1), x, n);
}
return -1;
}
```

```
    /* if x is present in arr[] then returns the
    index of LAST occurrence of x in arr[0..n-1],
    otherwise returns -1 */
    static int last(int []arr, int low,
                        int high, int x, int n)
    {
    if(high >= low)
    {
        /*low + (high - low)/2;*/
        int mid = (low + high)/2;
        if( ( mid == n-1 || x < arr[mid+1])
                            && arr[mid] == x )
        return mid;
        else if(x < arr[mid])
        return last(arr, low, (mid -1), x, n);
        else
        return last(arr, (mid + 1), high, x, n);
    }
    return -1;
    }

    public static void Main()
    {
        int []arr = {1, 2, 2, 3, 3, 3, 3};

        // Element to be counted in arr[]
        int x = 3;
        int n = arr.Length;
        int c = count(arr, x, n);

        Console.Write(x + " occurs " + c + " times");
    }
}
// This code is contributed by Sam007
```

Output:

```
3 occurs 4 times
```

Time Complexity: O(Logn)
Programming Paradigm: Divide & Conquer

Improved By : Sam007, ChitraNayal

Source

https://www.geeksforgeeks.org/count-number-of-occurrences-or-frequency-in-a-sorted-array/

Chapter 32

Count numbers with difference between number and its digit sum greater than specific value

Count numbers with difference between number and its digit sum greater than specific value - GeeksforGeeks

Given a positive value N, we need to find the count of numbers smaller than N such that the difference between the number and sum of its digits is greater than or equal to given specific value diff.
Examples:

```
Input : N = 13, diff = 2
Output : 4
Then 10, 11, 12 and 13 satisfy the given
condition shown below,
10 - sumofdigit(10) = 9 >= 2
11 - sumofdigit(11) = 9 >= 2
12 - sumofdigit(12) = 9 >= 2
13 - sumofdigit(13) = 9 >= 2
Whereas no number from 1 to 9 satisfies
above equation so final result will be 4
```

We can solve this problem by observing a fact that for a number k less than N,

```
if k - sumofdigit(k) >= diff then
above equation will be true for (k+1)
also because we know that sumofdigit(k+1)
is not greater than sumofdigit(k) + 1
```

```
so, k + 1 - sumofdigit(k + 1) >=
k - sumofdigit(k)
but we know that right side of above
inequality is greater than diff,
so left side will also be greater than
diff.
```

So, finally we can say that if a number k satisfies the difference condition then (k + 1) will also satisfy same equation so our job is to find the smallest number which satisfies the difference condition then all numbers greater than this and up to N will satisfy the condition so our answer will be N – smallest number we found.
We can find the smallest number satisfying this condition using binary search so total time complexity of solution will be O(log N)

C/C++

```
 /*  C++ program to count total numbers which
   have difference with sum of digits greater
   than specific value */
#include <bits/stdc++.h>
using namespace std;

//  Utility method to get sum of digits of K
int sumOfDigit(int K)
{
    //  loop until K is not zero
    int sod = 0;
    while (K)
    {
        sod += K % 10;
        K /= 10;
    }
    return sod;
}

// method returns count of numbers smaller than N,
// satisfying difference condition
int totalNumbersWithSpecificDifference(int N, int diff)
{
    int low = 1, high = N;

    //  binary search while loop
    while (low <= high)
    {
        int mid = (low + high) / 2;

        /* if difference between number and its sum
```

```
           of digit is smaller than given difference
           then  smallest number will be on left side */
        if (mid - sumOfDigit(mid) < diff)
            low = mid + 1;

        /* if difference between number and its sum
           of digit is greater than or equal to given
           difference then  smallest number will be on
           right side */
        else
            high = mid - 1;
    }

    // return the difference between 'smallest number
    // found' and 'N' as result
    return (N - high);
}

//  Driver code to test above methods
int main()
{
    int N = 13;
    int diff = 2;

    cout << totalNumbersWithSpecificDifference(N, diff);
    return 0;
}
```

Java

```
 /*  Java program to count total numbers which
   have difference with sum of digits greater
   than specific value */

class Test
{
    //  Utility method to get sum of digits of K
    static int sumOfDigit(int K)
    {
        //  loop until K is not zero
        int sod = 0;
        while (K != 0)
        {
            sod += K % 10;
            K /= 10;
        }
        return sod;
    }
```

```
    // method returns count of numbers smaller than N,
    // satisfying difference condition
    static int totalNumbersWithSpecificDifference(int N, int diff)
    {
        int low = 1, high = N;

        //  binary search while loop
        while (low <= high)
        {
            int mid = (low + high) / 2;

            /* if difference between number and its sum
               of digit is smaller than given difference
               then  smallest number will be on left side */
            if (mid - sumOfDigit(mid) < diff)
                low = mid + 1;

            /* if difference between number and its sum
               of digit is greater than or equal to given
               difference then  smallest number will be on
               right side */
            else
                high = mid - 1;
        }

        // return the difference between 'smallest number
        // found' and 'N' as result
        return (N - high);
    }

    // Driver method
    public static void main(String args[])
    {
        int N = 13;
        int diff = 2;

        System.out.println(totalNumbersWithSpecificDifference(N, diff));
    }
}
```

Python3

```
 # Python program to count total numbers which
# have difference with sum of digits greater
# than specific value
```

```
#  Utility method to get sum of digits of K
def sumOfDigit(K):

    #  loop until K is not zero
    sod = 0
    while (K):

        sod =sod + K % 10
        K =K // 10

    return sod


# method returns count of
# numbers smaller than N,
# satisfying difference condition
def totalNumbersWithSpecificDifference(N,diff):

    low = 1
    high = N

    #  binary search while loop
    while (low <= high):

        mid = (low + high) // 2

        ''' if difference between number and its sum
           of digit is smaller than given difference
           then  smallest number will be on left side'''
        if (mid - sumOfDigit(mid) < diff):
            low = mid + 1

          # if difference between number and its sum
          # of digit  greater than  equal to given
          # difference then  smallest number will be on
          # right side
        else:

            high = mid - 1


    # return the difference between 'smallest number
    # found' and 'N' as result
    return (N - high)

#  Driver code to test above methods
N = 13
diff = 2
```

```
print(totalNumbersWithSpecificDifference(N, diff))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to count total numbers
// which have difference with sum of
// digits greater than specific value
using System;

class Test {

    // Utility method to get sum
    // of digits of K
    static int sumOfDigit(int K)
    {

        // loop until K is not zero
        int sod = 0;
        while (K != 0)
        {
            sod += K % 10;
            K /= 10;
        }
        return sod;
    }

    // method returns count of numbers
    // smaller than N, satisfying
    // difference condition
    static int totalNumbersWithSpecificDifference(int N,
                                                  int diff)
    {
        int low = 1, high = N;

        // binary search while loop
        while (low <= high)
        {
            int mid = (low + high) / 2;

            // if difference between number and
            // its sum of digit is smaller than
            // given difference then smallest
            // number will be on left side
            if (mid - sumOfDigit(mid) < diff)
                low = mid + 1;
```

```
            // if difference between number and
            // its sum of digit is greater than
            // or equal to given difference then
            // smallest number will be on right side
            else
                high = mid - 1;
        }

        // return the difference between
        // 'smallest number found'
        // and 'N' as result
        return (N - high);
    }

    // Driver code
    public static void Main()
    {
        int N = 13;
        int diff = 2;

        Console.Write(totalNumbersWithSpecificDifference(N, diff));
    }
}

// This code is contributed by nitin mittal
```

PHP

```
 <?php
// PHP program to count total numbers which
// have difference with sum of digits greater
// than specific value

// method to get sum of digits of K
function sumOfDigit($K)
{

    // loop until K is not zero
    $sod = 0;
    while ($K)
    {
        $sod += $K % 10;
        $K /= 10;
    }
    return $sod;
}
```

```
// method returns count of
// numbers smaller than N,
// satisfying difference condition
function totalNumbersWithSpecificDifference($N, $diff)
{
    $low = 1; $high = $N;

    // binary search while loop
    while ($low <= $high)
    {
        $mid = floor(($low + $high) / 2);

        /* if difference between number and its sum
           of digit is smaller than given difference
           then smallest number will be on left side */
        if ($mid - sumOfDigit($mid) < $diff)
            $low = $mid + 1;

        /* if difference between number and its sum
           of digit is greater than or equal to given
           difference then smallest number will be on
           right side */
        else
            $high = $mid - 1;
    }

    // return the difference
    // between 'smallest number
    // found' and 'N' as result
    return ($N - $high);
}

// Driver Code
$N = 13;
$diff = 2;
echo totalNumbersWithSpecificDifference($N, $diff);

// This code is contributed by nitin mittal
?>
```

Output:

```
4
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/count-numbers-difference-number-digit-sum-greater-specific-value/

Chapter 33

Count of index pairs with equal elements in an array

Count of index pairs with equal elements in an array - GeeksforGeeks

Given an array of **n** elements. The task is to count the total number of indices (i, j) such that arr[i] = arr[j] and i != j

Examples :

```
Input : arr[] = {1, 1, 2}
Output : 1
As arr[0] = arr[1], the pair of indices is (0, 1)

Input : arr[] = {1, 1, 1}
Output : 3
As arr[0] = arr[1], the pair of indices is (0, 1),
(0, 2) and (1, 2)

Input : arr[] = {1, 2, 3}
Output : 0
```

Method 1 (Brute Force):
For each index i, find element after it with same value as arr[i]. Below is the implementation of this approach:

C++

```
 // C++ program to count of pairs with equal
// elements in an array.
#include<bits/stdc++.h>
```

```
using namespace std;

// Return the number of pairs with equal
// values.
int countPairs(int arr[], int n)
{
    int ans = 0;

    // for each index i and j
    for (int i = 0; i < n; i++)
        for (int j = i+1; j < n; j++)

            // finding the index with same
            // value but different index.
            if (arr[i] == arr[j])
                ans++;
    return ans;
}

// Driven Program
int main()
{
    int arr[] = { 1, 1, 2 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << countPairs(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java program to count of pairs with equal
// elements in an array.
class GFG {

    // Return the number of pairs with equal
    // values.
    static int countPairs(int arr[], int n)
    {
        int ans = 0;

        // for each index i and j
        for (int i = 0; i < n; i++)
            for (int j = i+1; j < n; j++)

                // finding the index with same
                // value but different index.
                if (arr[i] == arr[j])
                    ans++;
```

```
        return ans;
    }

    //driver code
    public static void main (String[] args)
    {
        int arr[] = { 1, 1, 2 };
        int n = arr.length;

        System.out.println(countPairs(arr, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to
# count of pairs with equal
# elements in an array.

# Return the number of
# pairs with equal values.
def countPairs(arr, n):

    ans = 0

    # for each index i and j
    for i in range(0 , n):
        for j in range(i + 1, n):

            # finding the index
            # with same value but
            # different index.
            if (arr[i] == arr[j]):
                ans += 1
    return ans

# Driven Code
arr = [1, 1, 2 ]
n = len(arr)
print(countPairs(arr, n))

# This code is contributed
# by Smitha
```

C#

```
 // C# program to count of pairs with equal
// elements in an array.
using System;

class GFG {

    // Return the number of pairs with equal
    // values.
    static int countPairs(int []arr, int n)
    {
        int ans = 0;

        // for each index i and j
        for (int i = 0; i < n; i++)
            for (int j = i+1; j < n; j++)

                // finding the index with same
                // value but different index.
                if (arr[i] == arr[j])
                    ans++;
        return ans;
    }

    // Driver code
    public static void Main ()
    {
        int []arr = { 1, 1, 2 };
        int n = arr.Length;

        Console.WriteLine(countPairs(arr, n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to count of
// pairs with equal elements
// in an array.

// Return the number of pairs
// with equal values.
function countPairs( $arr, $n)
{
    $ans = 0;
```

```
    // for each index i and j
    for ( $i = 0; $i < $n; $i++)
        for ( $j = $i + 1; $j < $n; $j++)

            // finding the index with same
            // value but different index.
            if ($arr[$i] == $arr[$j])
                $ans++;
    return $ans;
}

// Driven Code
$arr = array( 1, 1, 2 );
$n = count($arr);
echo countPairs($arr, $n) ;

// This code is contributed by anuj_67.
?>
```

Output :

```
1
```

Time Complexity : $O(n^2)$

Method 2 (Efficient approach):
The idea is to count the frequency of each number and then find the number of pairs with equal elements. Suppose, a number x appears k times at index i_1, i_2,….,i_k. Then pick any two indexes i_x and i_y which will be counted as 1 pair. Similarly, i_y and i_x can also be pair. So, choose ${}^nC_2$ is the number of pairs such that arr[i] = arr[j] = x.

Below is the implementation of this approach:

C++

```
 // C++ program to count of index pairs with
// equal elements in an array.
#include<bits/stdc++.h>
using namespace std;

// Return the number of pairs with equal
// values.
int countPairs(int arr[], int n)
{
    unordered_map<int, int> mp;
```

```
    // Finding frequency of each number.
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // Calculating pairs of each value.
    int ans = 0;
    for (auto it=mp.begin(); it!=mp.end(); it++)
    {
        int count = it->second;
        ans += (count * (count - 1))/2;
    }

    return ans;
}

// Driven Program
int main()
{
    int arr[] = {1, 1, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << countPairs(arr, n) << endl;
    return 0;
}
```

Output :

```
1
```

Time Complexity : O(n)

Source:
http://stackoverflow.com/questions/26772364/efficient-algorithm-for-counting-number-of-pairs-of-identical-elemen
comment42124861_26772516

Improved By : ash_maurya, vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/count-index-pairs-equal-elements-array/

Chapter 34

Count of only repeated element in a sorted array of consecutive elements

Count of only repeated element in a sorted array of consecutive elements - GeeksforGeeks

Given a sorted array of consecutive elements. The array has only one element repeated many times. The task is to find length of the sequence of repeated element.

Expected Time Complexity : Less than 0(n)

Examples:

```
Input  : arr[] = {1, 2, 3, 4, 4, 4, 5, 6}
Output : 4 3
Repeated element is 4 and it appears 3 times.

Input  : arr[] = {4, 4, 4, 4, 4}
Output : 4 5

Input  : arr[] = {6, 7, 8, 9, 10, 10, 11}
Output : 10 2

Input  : arr[] = {6, 7, 8, 9, 10, 10, 10}
Output : 10 3
```

We need to find two things:

1. Number of times the element repeats. If the array is sorted and if max-difference of two adjacent elements is 1, then the length of the repeated sequence is n – 1 – (array[n-1] – array[0])

2. Value of the element. To the value, we do Binary Search.

C++

```
 // C++ program to find the only repeated element
// and number of times it appears
#include <bits/stdc++.h>
using namespace std;

// Assumptions : vector a is sorted, max-difference
// of two adjacent elements is 1
pair<int, int> sequence(const vector<int>& a)
{
    if (a.size() == 0)
        return {0, 0};

    int s = 0;
    int e = a.size() - 1;
    while (s < e)
    {
        int m = (s + e) / 2;

        // if a[m] = m + a[0], there is no
        // repeating character in [s..m]
        if (a[m] >= m + a[0])
            s = m + 1;

       // if a[m] < m + a[0], there is a
       // repeating character in [s..m]
        else
            e = m;
    }
    return {a[s], a.size() - (a[a.size() - 1] - a[0])};
}

// Driver code
int main()
{
    pair<int, int>p = sequence({ 1, 2, 3, 4, 4, 4, 5, 6 });
    cout << "Repeated element is " << p.first
         << ", it appears " << p.second << " times";
    return 0;
}
```

Java

```
 // Java program to find the only repeated element
```

```
// and number of times it appears

import java.awt.Point;
import java.util.Arrays;
import java.util.Vector;

class Test
{
    // Assumptions : vector a is sorted, max-difference
    // of two adjacent elements is 1
    static Point sequence(Vector<Integer> a)
    {
        if (a.size() == 0)
            return new Point(0, 0);

        int s = 0;
        int e = a.size() - 1;
        while (s < e)
        {
            int m = (s + e) / 2;

            // if a[m] = m + a[0], there is no
            // repeating character in [s..m]
            if (a.get(m) >= m + a.get(0))
                s = m + 1;

           // if a[m] < m + a[0], there is a
           // repeating character in [s..m]
            else
                e = m;
        }
        return new Point(a.get(s), a.size() - (a.get(a.size() - 1) - a.get(0)));
    }

    // Driver method
    public static void main(String args[])
    {
        Integer array[] = new Integer[]{1, 2, 3, 4, 4, 4, 5, 6};
        Point p = sequence(new Vector<>(Arrays.asList(array)));
        System.out.println("Repeated element is " + p.x +
                           ", it appears " + p.y + " times");
    }
}
```

Output:

```
Repeated element is 4, it appears 3 times
```

Source

https://www.geeksforgeeks.org/count-of-only-repeated-element-in-a-sorted-array-of-consecutive-elements/

Chapter 35

Count pairs from two sorted matrices with given sum

Count pairs from two sorted matrices with given sum - GeeksforGeeks

Given two sorted matrices **mat1** and **mat2** of size **n x n** of distinct elements. Given a value **x**. The problem is to count all pairs from both matrices whose sum is equal to **x**.
Note: The pair has an element from each matrix. Matrices are strictly sorted which means that matrices are sorted in a way such that all elements in a row are sorted in increasing order and for row 'i', where 1 <= i <= n-1, first element of row 'i' is greater than the last element of row 'i-1'.

Examples:

```
Input : mat1[][] = { {1, 5, 6},
                     {8, 10, 11},
                    {15, 16, 18} }

    mat2[][] = { {2, 4, 7},
                 {9, 10, 12},
                 {13, 16, 20} }
       x = 21
Output : 4
The pairs are:
(1, 20), (5, 16), (8, 13) and (11, 10).
```

Method 1 (Naive Approach): For each element **ele** of mat1[][] linearly search **(x − ele)** in mat2[][].

C++

```
// C++ implementation to count pairs from two
```

```
// sorted matrices whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function to search 'val' in mat[][]
// returns true if 'val' is present
// else false
bool valuePresent(int mat[][SIZE], int n, int val)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (mat[i][j] == val)

                // 'val' found
                return true;

    // 'val' not found
    return false;
}

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[][SIZE], int mat2[][SIZE],
               int n, int x)
{
    int count = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)

            // if value (x-mat1[i][j]) is found in mat2[][]
            if (valuePresent(mat2, n, x - mat1[i][j]))
                count++;

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    int mat1[][SIZE] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };
```

```
    int mat2[][SIZE] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    cout << "Count = "
         << countPairs(mat1, mat2, n, x);

    return 0;
}
```

Java

```
 // java implementation to count
// pairs from twosorted matrices
// whose sum is equal to a given value
import java.io.*;

class GFG
{
    int SIZE= 10;

    // function to search 'val' in mat[][]
    // returns true if 'val' is present
    // else false
    static boolean valuePresent(int mat[][], int n,
                                            int val)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (mat[i][j] == val)

                    // 'val' found
                    return true;

        // 'val' not found
        return false;
    }

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int mat1[][], int mat2[][],
                                        int n, int x)
    {
```

```
        int count = 0;

        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
            {
                // if value (x-mat1[i][j]) is
                // found in mat2[][]
                if (valuePresent(mat2, n, x - mat1[i][j]))
                   count++;
            }
        // required count of pairs
        return count;
    }

    // Driver program
    public static void main (String[] args)
    {
        int mat1[][] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };

        int mat2[][] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

        int n = 3;
        int x = 21;

        System.out.println ("Count = " +
                            countPairs(mat1, mat2, n, x));

    }
}

// This article is contributed by vt_m
```

Python3

```
 # Python3 implementation to count pairs
# from two sorted matrices whose sum is
# equal to a given value x

# function to search 'val' in mat[][]
# returns true if 'val' is present else
# false
def valuePresent(mat, n, val):

    for i in range(0, n):
```

```
        for j in range(0, n):

            if mat[i][j] == val:

                # 'val' found
                return True

    # 'val' not found
    return False

# function to count pairs from two sorted
# matrices whose sum is equal to a given
# value x
def countPairs(mat1, mat2, n, x):

    count = 0

    for i in range(0, n):
        for j in range(0, n):

            # if value (x-mat1[i][j]) is found
            # in mat2[][]
            if valuePresent(mat2, n, x - mat1[i][j]):
                count += 1

    # required count of pairs
    return count

# Driver program
mat1 = [[ 1, 5, 6 ],
        [ 8, 10, 11 ],
        [ 15, 16, 18 ] ]

mat2 = [ [ 2, 4, 7 ],
         [ 9, 10, 12 ],
         [ 13, 16, 20 ] ]

n = 3
x = 21

print( "Count = "),
print(countPairs(mat1, mat2, n, x))

# This code is contributed by upendra bartwal
```

C#

```
 //C# implementation to count
```

```
// pairs from twosorted matrices
// whose sum is equal to a given value
using System;

class GFG
{
    // int SIZE= 10;

    // function to search 'val' in mat[][]
    // returns true if 'val' is present
    // else false
    static bool valuePresent(int[,] mat, int n,
                                        int val)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (mat[i, j] == val)

                    // 'val' found
                    return true;

        // 'val' not found
        return false;
    }

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int [,]mat1, int [,]mat2,
                                        int n, int x)
    {
        int count = 0;

        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
            {
                // if value (x-mat1[i][j]) is
                // found in mat2[][]
                if (valuePresent(mat2, n, x - mat1[i,j]))
                count++;
            }
        // required count of pairs
        return count;
    }

    // Driver program
    public static void Main ()
    {
```

```
        int [,]mat1 = { { 1, 5, 6 },
                        { 8, 10, 11 },
                        { 15, 16, 18 } };

        int [,]mat2 = { { 2, 4, 7 },
                        { 9, 10, 12 },
                        { 13, 16, 20 } };

        int n = 3;
        int x = 21;

        Console.WriteLine("Count = " +
                        countPairs(mat1, mat2, n, x));

    }
}

// This article is contributed by vt_m
```

Output:

```
Count = 4
```

Time Complexity: O(n^4).
Auxiliary Space: O(1).

Method 2 (Binary Search): As matrix is strictly sorted, use the concept of binary search technique. For each element **ele** of mat1[][] apply the binary search technique on the elements of the first column of mat2[][] to find the row index number of the largest element smaller than equal to **(x − ele)**. Let it be **row_no**. If no such row exists then no pair can be formed with element **ele**. Else apply the concept of binary search technique to find the value **(x − ele)** in the row represented by **row_no** in mat2[][]. If value found then increment **count**.

C++

```
 // C++ implementation to count pairs from two
// sorted matrices whose sum is equal to a given
// value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function returns the row index no of largest
// element smaller than equal to 'x' in first
```

```
// column of mat[][]. If no such element exists
// then it returns -1.
int binarySearchOnRow(int mat[SIZE][SIZE],
                      int l, int h, int x)
{
    while (l <= h) {
        int mid = (l + h) / 2;

        // if 'x' is greater than or equal to mat[mid][0],
        // then search in mat[mid+1...h][0]
        if (mat[mid][0] <= x)
            l = mid + 1;

        // else search in mat[l...mid-1][0]
        else
            h = mid - 1;
    }

    // required row index number
    return h;
}

// function to search 'val' in mat[row][]
bool binarySearchOnCol(int mat[][SIZE], int l, int h,
                       int val, int row)
{
    while (l <= h) {
        int mid = (l + h) / 2;

        // 'val' found
        if (mat[row][mid] == val)
            return true;

        // search in mat[row][mid+1...h]
        else if (mat[row][mid] < val)
            l = mid + 1;

        // search in mat[row][l...mid-1]
        else
            h = mid - 1;
    }

    // 'val' not found
    return false;
}

// function to search 'val' in mat[][]
// returns true if 'val' is present
```

```
// else false
bool searchValue(int mat[][SIZE],
                 int n, int val)
{
    // to get the row index number of the largest element
    // smaller than equal to 'val' in mat[][]
    int row_no = binarySearchOnRow(mat, 0, n - 1, val);

    // if no such row exists, then
    // 'val' is not present
    if (row_no == -1)
        return false;

    // to search 'val' in mat[row_no][]
    return binarySearchOnCol(mat, 0, n - 1, val, row_no);
}

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[][SIZE], int mat2[][SIZE],
               int n, int x)
{
    int count = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            // if value (x-mat1[i][j]) is found in mat2[][]
            if (searchValue(mat2, n, x - mat1[i][j]))
                count++;

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    int mat1[][SIZE] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };

    int mat2[][SIZE] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

    int n = 3;
    int x = 21;
```

```
    cout << "Count = "
         << countPairs(mat1, mat2, n, x);

    return 0;
}
```

Java

```
 // java implementation to count
// pairs from two sorted matrices
// whose sum is equal to a given
// value x
import java.io.*;

class GFG {
    int SIZE= 10;

    // function returns the row index no of largest
    // element smaller than equal to 'x' in first
    // column of mat[][]. If no such element exists
    // then it returns -1.
    static int binarySearchOnRow(int mat[][], int l,
                                      int h, int x)
    {
        while (l <= h)
        {
            int mid = (l + h) / 2;

            // if 'x' is greater than or
            // equal to mat[mid][0], then
            // search in mat[mid+1...h][0]
            if (mat[mid][0] <= x)
                l = mid + 1;

            // else search in mat[l...mid-1][0]
            else
                h = mid - 1;
        }

        // required row index number
        return h;
    }

    // function to search 'val' in mat[row][]
    static boolean binarySearchOnCol(int mat[][], int l, int h,
                                           int val, int row)
    {
        while (l <= h)
```

```
    {
        int mid = (l + h) / 2;

        // 'val' found
        if (mat[row][mid] == val)
            return true;

        // search in mat[row][mid+1...h]
        else if (mat[row][mid] < val)
            l = mid + 1;

        // search in mat[row][l...mid-1]
        else
            h = mid - 1;
    }

    // 'val' not found
    return false;
}

// function to search 'val' in mat[][]
// returns true if 'val' is present
// else false
static boolean searchValue(int mat[][],
                           int n, int val)
{
    // to get the row index number
    // of the largest element smaller
    // than equal to 'val' in mat[][]
    int row_no = binarySearchOnRow(mat, 0, n - 1, val);

    // if no such row exists, then
    // 'val' is not present
    if (row_no == -1)
        return false;

    // to search 'val' in mat[row_no][]
    return binarySearchOnCol(mat, 0, n - 1, val, row_no);
}

// function to count pairs from
// two sorted matrices whose sum
// is equal to a given value x
static int countPairs(int mat1[][], int mat2[][],
                                    int n, int x)
{
    int count = 0;
```

```
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
            {
                // if value (x-mat1[i][j]) is found in mat2[][]
                if (searchValue(mat2, n, x - mat1[i][j]))
                    count++;
            }
        // required count of pairs
        return count;
    }

    // Driver program
    public static void main (String[] args)
    {
        int mat1[][] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };

        int mat2[][] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

        int n = 3;
        int x = 21;

        System.out.println ( "Count = "  +
                           countPairs(mat1, mat2, n, x));

    }
}

// This code is contributed by vt_m
```

C#

```
 // C# implementation to count
// pairs from two sorted matrices
// whose sum is equal to a given
// value x
using System;

class GFG
{
    //int SIZE= 10;

    // function returns the row index no of largest
    // element smaller than equal to 'x' in first
    // column of mat[][]. If no such element exists
```

```
// then it returns -1.
static int binarySearchOnRow(int [,]mat, int l,
                                int h, int x)
{
    while (l <= h)
    {
        int mid = (l + h) / 2;

        // if 'x' is greater than or
        // equal to mat[mid][0], then
        // search in mat[mid+1...h][0]
        if (mat[mid,0] <= x)
            l = mid + 1;

        // else search in mat[l...mid-1][0]
        else
            h = mid - 1;
    }

    // required row index number
    return h;
}

// function to search 'val' in mat[row][]
static bool binarySearchOnCol(int [,]mat, int l, int h,
                                        int val, int row)
{
    while (l <= h)
    {
        int mid = (l + h) / 2;

        // 'val' found
        if (mat[row,mid] == val)
            return true;

        // search in mat[row][mid+1...h]
        else if (mat[row,mid] < val)
            l = mid + 1;

        // search in mat[row][l...mid-1]
        else
            h = mid - 1;
    }

    // 'val' not found
    return false;
}
```

```
// function to search 'val' in mat[][]
// returns true if 'val' is present
// else false
static bool searchValue(int [,]mat,
                        int n, int val)
{
    // to get the row index number
    // of the largest element smaller
    // than equal to 'val' in mat[][]
    int row_no = binarySearchOnRow(mat, 0, n - 1, val);

    // if no such row exists, then
    // 'val' is not present
    if (row_no == -1)
        return false;

    // to search 'val' in mat[row_no][]
    return binarySearchOnCol(mat, 0, n - 1, val, row_no);
}

// function to count pairs from
// two sorted matrices whose sum
// is equal to a given value x
static int countPairs(int [,]mat1, int [,]mat2,
                                    int n, int x)
{
    int count = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            // if value (x-mat1[i][j]) is found in mat2[][]
            if (searchValue(mat2, n, x - mat1[i,j]))
                count++;
        }
    // required count of pairs
    return count;
}

// Driver program
public static void Main ()
{
    int [,]mat1 = { { 1, 5, 6 },
                    { 8, 10, 11 },
                    { 15, 16, 18 } };

    int [,]mat2 = { { 2, 4, 7 },
                    { 9, 10, 12 },
```

```
                        { 13, 16, 20 } };

        int n = 3;
        int x = 21;

        Console.WriteLine ( "Count = " +
                        countPairs(mat1, mat2, n, x));

    }
}

// This code is contributed by vt_m
```

Output:

```
Count = 4
```

Time Complexity: ($n^2 \log_2 n$).
Auxiliary Space: O(1).

Method 3 (Hashing): Create a hash table and insert all the elements of mat2[][] in it. Now for each element **ele** of mat1[][] find **(x − ele)** in the hash table.

```
 // C++ implementation to count pairs from two
// sorted matrices whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[][SIZE], int mat2[][SIZE],
               int n, int x)
{
    int count = 0;

    // unordered_set 'us' implemented as hash table
    unordered_set<int> us;

    // insert all the elements of mat2[][] in 'us'
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            us.insert(mat2[i][j]);
```

```
    // for each element of mat1[][]
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)

            // if (x-mat1[i][j]) is in 'us'
            if (us.find(x - mat1[i][j]) != us.end())
                count++;

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    int mat1[][SIZE] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };

    int mat2[][SIZE] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    cout << "Count = "
         << countPairs(mat1, mat2, n, x);

    return 0;
}
```

Output:

```
Count = 4
```

Time complexity: $O(n^2)$.
Auxiliary Space: $O(n^2)$.

Method 4 (Efficient Approach): From the top leftmost element traverse mat1[][] in forward direction (i.e., from the topmost row up to last, each row is being traversed from left to right) and from the bottom rightmost element traverse mat2[][] in backward direction (i.e, from the bottom row up to first, each row is being traversed from right to left). For each element **e1** of mat1[][] and **e2** of mat2[][] encountered, calculate **val** = (e1 + e2). If val == x, increment **count**. Else if val is less than x, move to next element of mat1[][] in forward direction. Else move to next element of mat2[][] in backward direction. Continue

this process until either of the two matrices gets completely traversed.

C++

```
 // C++ implementation to count pairs from two
// sorted matrices whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[][SIZE], int mat2[][SIZE],
                                    int n, int x)
{
    // 'r1' and 'c1' for pointing current element
    // of mat1[][]
    // 'r2' and 'c2' for pointing current element
    // of mat2[][]
    int r1 = 0, c1 = 0;
    int r2 = n - 1, c2 = n - 1;

    // while there are more elements
    // in both the matrices
    int count = 0;
    while ((r1 < n) && (r2 >= -1)) {
        int val = mat1[r1][c1] + mat2[r2][c2];

        // if true
        if (val == x) {

            // increment 'count'
            count++;

            // move mat1[][] column 'c1' to right
            // move mat2[][] column 'c2' to left
            c1++;
            c2--;
        }

        // if true, move mat1[][] column 'c1' to right
        else if (val < x)
            c1++;

        // else move mat2[][] column 'c2' to left
```

```
        else
            c2--;

        // if 'c1' crosses right boundary
        if (c1 == n) {

            // reset 'c1'
            c1 = 0;

            // increment row 'r1'
            r1++;
        }

        // if 'c2' crosses left boundary
        if (c2 == -1) {

            // reset 'c2'
            c2 = n - 1;

            // decrement row 'r2'
            r2--;
        }
    }

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    int mat1[][SIZE] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };

    int mat2[][SIZE] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    cout << "Count = "
         << countPairs(mat1, mat2, n, x);

    return 0;
}
```

Java

```
 // java implementation to count
// pairs from two  sorted
// matrices whose sum is
// equal to agiven value x
import java.io.*;

class GFG
{
    int SIZE = 10;

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int mat1[][], int mat2[][],
                                        int n, int x)
    {
        // 'r1' and 'c1' for pointing current
        // element of mat1[][]
        // 'r2' and 'c2' for pointing current
        // element of mat2[][]
        int r1 = 0, c1 = 0;
        int r2 = n - 1, c2 = n - 1;

        // while there are more elements
        // in both the matrices
        int count = 0;
        while ((r1 < n) && (r2 >= -1))
        {
            int val = mat1[r1][c1] + mat2[r2][c2];

            // if true
            if (val == x) {

                // increment 'count'
                count++;

                // move mat1[][] column 'c1' to right
                // move mat2[][] column 'c2' to left
                c1++;
                c2--;
            }

            // if true, move mat1[][]
            // column 'c1' to right
            else if (val < x)
                c1++;
```

```
        // else move mat2[][] column
        // 'c2' to left
        else
            c2--;

        // if 'c1' crosses right boundary
        if (c1 == n) {

            // reset 'c1'
            c1 = 0;

            // increment row 'r1'
            r1++;
        }

        // if 'c2' crosses left boundary
        if (c2 == -1) {

            // reset 'c2'
            c2 = n - 1;

            // decrement row 'r2'
            r2--;
        }
    }

    // required count of pairs
    return count;
}

// Driver code
public static void main (String[] args)
{
    int mat1[][] = { { 1, 5, 6 },
                     { 8, 10, 11 },
                     { 15, 16, 18 } };

    int mat2[][] = { { 2, 4, 7 },
                     { 9, 10, 12 },
                     { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    System.out.println ( "Count = " +
                         countPairs(mat1, mat2, n, x));
```

```
    }
}

// This article is contributed by vt_m
```

C#

```
 // C# implementation to count pairs
// from two sorted matrices whose
// sum is equal to a given value x
using System;

class GFG {

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int [,]mat1,
            int [,]mat2, int n, int x)
    {

        // 'r1' and 'c1' for pointing
        // current element of mat1[][]
        // 'r2' and 'c2' for pointing
        // current element of mat2[][]
        int r1 = 0, c1 = 0;
        int r2 = n - 1, c2 = n - 1;

        // while there are more elements
        // in both the matrices
        int count = 0;
        while ((r1 < n) && (r2 >= -1))
        {
            int val = mat1[r1,c1]
                        + mat2[r2,c2];

            // if true
            if (val == x) {

                // increment 'count'
                count++;

                // move mat1[][] column
                // 'c1' to right
                // move mat2[][] column
                // 'c2' to left
                c1++;
                c2--;
```

```
        }

        // if true, move mat1[][]
        // column 'c1' to right
        else if (val < x)
            c1++;

        // else move mat2[][] column
        // 'c2' to left
        else
            c2--;

        // if 'c1' crosses right
        // boundary
        if (c1 == n) {

            // reset 'c1'
            c1 = 0;

            // increment row 'r1'
            r1++;
        }

        // if 'c2' crosses left
        // boundary
        if (c2 == -1) {

            // reset 'c2'
            c2 = n - 1;

            // decrement row 'r2'
            r2--;
        }
    }

    // required count of pairs
    return count;
}

// Driver code
public static void Main ()
{
    int [,]mat1 = { { 1, 5, 6 },
                    { 8, 10, 11 },
                    { 15, 16, 18 } };

    int [,]mat2 = { { 2, 4, 7 },
                    { 9, 10, 12 },
```

```
                        { 13, 16, 20 } };

        int n = 3;
        int x = 21;

        Console.Write ( "Count = " +
            countPairs(mat1, mat2, n, x));

    }
}

// This code is contributed by
// nitin mittal
```

Output:

```
Count = 4
```

Time Complexity: O(n^2).
Auxiliary Space: O(1).

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/count-pairs-two-sorted-matrices-given-sum/

Chapter 36

Count pairs in a binary tree whose sum is equal to a given value x

Count pairs in a binary tree whose sum is equal to a given value x - GeeksforGeeks

Given a binary tree containing **n** distinct numbers and a value **x**. The problem is to count pairs in the given binary tree whose sum is equal to the given value **x**.

Examples:

```
Input :
        5
       / \
      3   7
     / \ / \
    2  4 6  8

        x = 10

Output : 3
The pairs are (3, 7), (2, 8) and (4, 6).
```

Naive Approach: One by one get each node of the binary tree through any of the tree traversals method. Pass the node say **temp**, the **root** of the tree and value **x** to another function say **findPair()**. In the function with the help of the **root** pointer traverse the tree again. One by one sum up these nodes with **temp** and check whether sum == x. If so, increment **count**. Calculate count = count / 2 as a single pair has been counted twice by the aforementioned method.

```
// C++ implementation to count pairs in a binary tree
```

```
// whose sum is equal to given value x
#include <bits/stdc++.h>
using namespace std;

// structure of a node of a binary tree
struct Node {
    int data;
    Node *left, *right;
};

// function to create and return a node
// of a binary tree
Node* getNode(int data)
{
    // allocate space for the node
    Node* new_node = (Node*)malloc(sizeof(Node));

    // put in the data
    new_node->data = data;
    new_node->left = new_node->right = NULL;
}

// returns true if a pair exists with given sum 'x'
bool findPair(Node* root, Node* temp, int x)
{
    // base case
    if (!root)
        return false;

    // pair exists
    if (root != temp && ((root->data + temp->data) == x))
        return true;

    // find pair in left and right subtress
    if (findPair(root->left, temp, x) || findPair(root->right, temp, x))
        return true;

    // pair does not exists with given sum 'x'
    return false;
}

// function to count pairs in a binary tree
// whose sum is equal to given value x
void countPairs(Node* root, Node* curr, int x, int& count)
{
    // if tree is empty
    if (!curr)
        return;
```

```
    // check whether pair exists for current node 'curr'
    // in the binary tree that sum up to 'x'
    if (findPair(root, curr, x))
        count++;

    // recursively count pairs in left subtree
    countPairs(root, curr->left, x, count);

    // recursively count pairs in right subtree
    countPairs(root, curr->right, x, count);
}

// Driver program to test above
int main()
{
    // formation of binary tree
    Node* root = getNode(5); /*        5      */
    root->left = getNode(3); /*       / \      */
    root->right = getNode(7); /*    3   7      */
    root->left->left = getNode(2); /*   / \ / \   */
    root->left->right = getNode(4); /*   2 4 6 8   */
    root->right->left = getNode(6);
    root->right->right = getNode(8);

    int x = 10;
    int count = 0;

    countPairs(root, root, x, count);
    count = count / 2;

    cout << "Count = " << count;

    return 0;
}
```

Output:

```
Count = 3
```

Time Complexity: O(n^2).

Efficient Approach: Following are the steps:

1. Convert given binary tree to doubly linked list. Refer this post.
2. Sort the doubly linked list obtained in Step 1. Refer this post.
3. Count Pairs in sorted doubly linked with sum equal to 'x'. Refer this post.

4. Display the count obtained in Step 4.

```
 // C++ implementation to count pairs in a binary tree
// whose sum is equal to given value x
#include <bits/stdc++.h>
using namespace std;

// structure of a node of a binary tree
struct Node {
    int data;
    Node *left, *right;
};

// function to create and return a node
// of a binary tree
Node* getNode(int data)
{
    // allocate space for the node
    Node* new_node = (Node*)malloc(sizeof(Node));

    // put in the data
    new_node->data = data;
    new_node->left = new_node->right = NULL;
}

// A simple recursive function to convert a given
// Binary tree to Doubly Linked List
// root     --> Root of Binary Tree
// head_ref --> Pointer to head node of created
// doubly linked list
void BToDLL(Node* root, Node** head_ref)
{
    // Base cases
    if (root == NULL)
        return;

    // Recursively convert right subtree
    BToDLL(root->right, head_ref);

    // insert root into DLL
    root->right = *head_ref;

    // Change left pointer of previous head
    if (*head_ref != NULL)
        (*head_ref)->left = root;

    // Change head of Doubly linked list
    *head_ref = root;
```

```
    // Recursively convert left subtree
    BToDLL(root->left, head_ref);
}

// Split a doubly linked list (DLL) into 2 DLLs of
// half sizes
Node* split(Node* head)
{
    Node *fast = head, *slow = head;
    while (fast->right && fast->right->right) {
        fast = fast->right->right;
        slow = slow->right;
    }
    Node* temp = slow->right;
    slow->right = NULL;
    return temp;
}

// Function to merge two sorted doubly linked lists
Node* merge(Node* first, Node* second)
{
    // If first linked list is empty
    if (!first)
        return second;

    // If second linked list is empty
    if (!second)
        return first;

    // Pick the smaller value
    if (first->data < second->data) {
        first->right = merge(first->right, second);
        first->right->left = first;
        first->left = NULL;
        return first;
    }
    else {
        second->right = merge(first, second->right);
        second->right->left = second;
        second->left = NULL;
        return second;
    }
}

// Function to do merge sort
Node* mergeSort(Node* head)
{
```

```
    if (!head || !head->right)
        return head;
    Node* second = split(head);

    // Recur for left and right halves
    head = mergeSort(head);
    second = mergeSort(second);

    // Merge the two sorted halves
    return merge(head, second);
}

// Function to count pairs in a sorted doubly linked list
// whose sum equal to given value x
int pairSum(Node* head, int x)
{
    // Set two pointers, first to the beginning of DLL
    // and second to the end of DLL.
    Node* first = head;
    Node* second = head;
    while (second->right != NULL)
        second = second->right;

    int count = 0;

    // The loop terminates when either of two pointers
    // become NULL, or they cross each other (second->right
    // == first), or they become same (first == second)
    while (first != NULL && second != NULL && first != second && second->right != first) {
        // pair found
        if ((first->data + second->data) == x) {
            count++;

            // move first in forward direction
            first = first->right;

            // move second in backward direction
            second = second->left;
        }
        else {
            if ((first->data + second->data) < x)
                first = first->right;
            else
                second = second->left;
        }
    }

    return count;
```

```
}

// function to count pairs in a binary tree
// whose sum is equal to given value x
int countPairs(Node* root, int x)
{
    Node* head = NULL;
    int count = 0;

    // Convert binary tree to
    // doubly linked list
    BToDLL(root, &head);

    // sort DLL
    head = mergeSort(head);

    // count pairs
    return pairSum(head, x);
}

// Driver program to test above
int main()
{
    // formation of binary tree
    Node* root = getNode(5); /*        5      */
    root->left = getNode(3); /*       / \      */
    root->right = getNode(7); /*    3   7      */
    root->left->left = getNode(2); /*   / \ / \   */
    root->left->right = getNode(4); /*   2 4 6 8   */
    root->right->left = getNode(6);
    root->right->right = getNode(8);

    int x = 10;

    cout << "Count = "
         << countPairs(root, x);

    return 0;
}
```

Output:

```
Count = 3
```

Time Complexity: O(nLog n).

Source

https://www.geeksforgeeks.org/count-pairs-in-a-binary-tree-whose-sum-is-equal-to-a-given-value-x/

Chapter 37

Count pairs in an array that hold i*arr[i] > j*arr[j]

Count pairs in an array that hold i*arr[i] > j*arr[j] - GeeksforGeeks

Given an array of integers arr[0..n-1], count all pairs (arr[i], arr[j]) in the such that i*arr[i] > j*arr[j], 0 =< i < j < n.

Examples :

```
Input : arr[] = {5 , 0, 10, 2, 4, 1, 6}
Output: 5
 Pairs which hold condition i*arr[i] > j*arr[j]
 are (10, 2) (10, 4) (10, 1) (2, 1) (4, 1)

Input  : arr[] = {8, 4, 2, 1}
Output : 2
```

A **Simple solution** is to run two loops. Pick each element of array one-by-one and for each element find element on right side of array that hold condition, then increment counter and last return counter value.
Below is the implementation of above idea:

C++

```
 // C++ program to count all pair that
// hold condition i*arr[i] > j*arr[j]
#include<iostream>
using namespace std;

// Return count of pair in given array
```

```
// such that  i*arr[i] > j*arr[j]
int CountPair(int arr[] , int n )
{
    int result = 0; // Initialize result

    for (int i=0; i<n; i++)
    {
        // Generate all pair and increment
        // counter if the hold given condition
        for (int j = i + 1; j < n; j++)
            if (i*arr[i] > j*arr[j] )
                result ++;
    }
    return result;
}

// Driver code
int main()
{
    int arr[] = {5 , 0, 10, 2, 4, 1, 6} ;
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Count of Pairs : "
         << CountPair(arr, n);
    return 0;
}
```

Java

```
 // Java Code for Count pairs in an
// array that hold i*arr[i] > j*arr[j]
class GFG {

    // Return count of pair in given array
    // such that  i*arr[i] > j*arr[j]
    public static int CountPair(int arr[] , int n )
    {
        int result = 0; // Initialize result

        for (int i = 0; i < n; i++)
        {
            // Generate all pair and increment
            // counter if the hold given condition
            for (int j = i + 1; j < n; j++)
                if (i*arr[i] > j*arr[j] )
                    result ++;
        }
        return result;
    }
```

```
    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int arr[] = {5 , 0, 10, 2, 4, 1, 6} ;
        int n = arr.length;
        System.out.println("Count of Pairs : " +
                           CountPair(arr, n));
    }
  }
// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # C# Code to Count pairs in an
# array that hold i*arr[i] > j*arr[j]

# Return count of pair in given array
# such that i*arr[i] > j*arr[j]
def CountPair(arr , n ):

    # Initialize result
    result = 0;

    for i in range (0, n):

        # Generate all pair and increment
        # counter if the hold given condition
        j = i + 1
        while(j < n):
            if (i * arr[i] > j * arr[j] ):
                result = result +1
            j = j + 1
    return result;

# Driver program to test above function */

arr = [5, 0, 10, 2, 4, 1, 6]
n = len(arr)
print("Count of Pairs : " , CountPair(arr, n))

# This code is contributed by Sam007.
```

C#

```
 // C# Code to Count pairs in an
// array that hold i*arr[i] > j*arr[j]
```

```
using System;

class GFG
{
    // Return count of pair in given array
    // such that i*arr[i] > j*arr[j]
    public static int CountPair(int []arr , int n )
    {
        // Initialize result
        int result = 0;

        for (int i = 0; i < n; i++)
        {
            // Generate all pair and increment
            // counter if the hold given condition
            for (int j = i + 1; j < n; j++)
                if (i*arr[i] > j*arr[j] )
                    result ++;
        }
        return result;
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int []arr = {5, 0, 10, 2, 4, 1, 6};
        int n = arr.Length;
        Console.WriteLine("Count of Pairs : " +
                          CountPair(arr, n));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to count all pair that
// hold condition i*arr[i] > j*arr[j]

// Return count of pair in given array
// such that i*arr[i] > j*arr[j]
function CountPair($arr , $n )
{

    // Initialize result
    $result = 0;
```

```
    for($i = 0; $i < $n; $i++)
    {

        // Generate all pair and increment
        // counter if the hold given condition
        for ($j = $i + 1; $j < $n; $j++)
            if ($i *  $arr[$i] > $j * $arr[$j] )
                $result ++;
    }
    return $result;
}

    // Driver code
    $arr = array(5, 0, 10, 2, 4, 1, 6) ;
    $n = sizeof($arr);
    echo "Count of Pairs : ",
    CountPair($arr, $n);

// This code is contributed by m_kit
?>
```

Output:

```
Count of Pairs : 5
```

Time Complexity: $O(n^2)$

An **efficient solution** of this problem takes O(n log n) time. The idea is based on an interesting fact about this problem that after modifying the array such that every element is multiplied with its index, this problem convert into Count Inversions in an array.
Algorithm :

```
Given an array 'arr' and it's size 'n'
1) First traversal array element, i goes from 0 to n-1
   a) Multiple each element with its index arr[i] = arr[i] * i
2) After that step 1. whole process is similar to Count Inversions in an array.
```

Below the implementation of above idea

C++

```
 // C++ program to count all pair that
// hold condition i*arr[i] > j*arr[j]
```

```
#include <bits/stdc++.h>
using namespace std;

/* This function merges two sorted arrays and
   returns inversion count in the arrays.*/
int merge(int arr[], int temp[], int left,
                      int mid, int right)
{
    int inv_count = 0;

    int i = left; /* index for left subarray*/
    int j = mid;  /* index for right subarray*/
    int k = left; /* ndex for resultant subarray*/
    while ((i <= mid - 1) && (j <= right))
    {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
        {
            temp[k++] = arr[j++];

            inv_count = inv_count + (mid - i);
        }
    }

    /* Copy the remaining elements of left
     subarray (if there are any) to temp*/
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    /* Copy the remaining elements of right
     subarray (if there are any) to temp*/
    while (j <= right)
        temp[k++] = arr[j++];

    /* Copy back the merged elements to original
      array*/
    for (i=left; i <= right; i++)
        arr[i] = temp[i];

    return inv_count;
}

/* An auxiliary recursive function that sorts
   the input array and returns the number of
   inversions in the array. */
int _mergeSort(int arr[], int temp[], int left,
                                      int right)
```

```
{
    int mid, inv_count = 0;
    if (right > left)
    {
        /* Divide the array into two parts and call
          _mergeSortAndCountInv() for each of
          the parts */
        mid = (right + left)/2;

        /* Inversion count will be sum of inversions in
           left-part, right-part and number of inversions
           in merging */
        inv_count  = _mergeSort(arr, temp, left, mid);
        inv_count += _mergeSort(arr, temp, mid+1, right);

        /*Merge the two parts*/
        inv_count += merge(arr, temp, left, mid+1, right);
    }

    return inv_count;
}

/* This function sorts the input array and
   returns the number of inversions in the
   array */
int countPairs(int arr[], int n)
{
    // Modify the array so that problem reduces to
    // count inversion problem.
    for (int i=0; i<n; i++)
        arr[i] = i*arr[i];

    // Count inversions using same logic as
    // below post
    // https://www.geeksforgeeks.org/counting-inversions/
    int temp[n];
    return _mergeSort(arr, temp, 0, n - 1);
}

// Driver code
int main()
{
    int arr[] = {5, 0, 10, 2, 4, 1, 6};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Count of Pairs : "
         << countPairs(arr, n);
    return 0;
}
```

Java

```
 // Java program to count all pair that
// hold condition i*arr[i] > j*arr[j]
import java.io.*;

class GFG
{
    // This function merges two sorted arrays and
    // returns inversion count in the arrays.
    static int merge(int arr[], int temp[], int left,
                                    int mid, int right)
    {
        int inv_count = 0;

        /* index for left subarray*/
        int i = left;

        /* index for right subarray*/
        int j = mid;
        /* ndex for resultant subarray*/
        int k = left;

        while ((i <= mid - 1) && (j <= right))
        {
            if (arr[i] <= arr[j])
                temp[k++] = arr[i++];
            else
            {
                temp[k++] = arr[j++];

                inv_count = inv_count + (mid - i);
            }
        }

        /* Copy the remaining elements of left
        subarray (if there are any) to temp*/
        while (i <= mid - 1)
            temp[k++] = arr[i++];

        /* Copy the remaining elements of right
        subarray (if there are any) to temp*/
        while (j <= right)
            temp[k++] = arr[j++];

        // Copy back the merged elements
        // to original array
        for (i = left; i <= right; i++)
```

```
        arr[i] = temp[i];

    return inv_count;
}

/* An auxiliary recursive function
that sorts the input array and
returns the number of inversions
in the array. */
static int _mergeSort(int arr[], int temp[],
                          int left,int right)
{
    int mid, inv_count = 0;
    if (right > left)
    {
        /* Divide the array into two parts and call
        _mergeSortAndCountInv() for each of
        the parts */
        mid = (right + left) / 2;

        // Inversion count will be sum of inversions in
        // left-part, right-part and number of inversions
        // in merging
        inv_count = _mergeSort(arr, temp, left, mid);
        inv_count += _mergeSort(arr, temp, mid+1, right);

        /*Merge the two parts*/
        inv_count += merge(arr, temp, left, mid+1, right);
    }

    return inv_count;
}

// This function sorts the input array and
// returns the number of inversions in the
// array
static int countPairs(int arr[], int n)
{
    // Modify the array so that problem reduces to
    // count inversion problem.
    for (int i = 0; i < n; i++)
        arr[i] = i * arr[i];

    // Count inversions using same logic as
    // below post
    // https://www.geeksforgeeks.org/counting-inversions/
    int temp[] = new int [n];
    return _mergeSort(arr, temp, 0, n - 1);
```

```
    }

    // Driver code

    public static void main (String[] args)
    {
        int arr[] = {5, 0, 10, 2, 4, 1, 6};
        int n = arr.length;
        System.out.print( "Count of Pairs : "
                          + countPairs(arr, n));

    }
}

// This code is contributed by vt_m
```

C#

```
 // C# program to count all pair that
// hold condition i*arr[i] > j*arr[j]
using System;

class GFG
{
    // This function merges two sorted arrays and
    // returns inversion count in the arrays.
    static int merge(int []arr, int []temp, int left,
                                int mid, int right)
    {
        int inv_count = 0;

        /* index for left subarray*/
        int i = left;

        /* index for right subarray*/
        int j = mid;
        /* ndex for resultant subarray*/
        int k = left;

        while ((i <= mid - 1) && (j <= right))
        {
            if (arr[i] <= arr[j])
                temp[k++] = arr[i++];
            else
            {
                temp[k++] = arr[j++];

                inv_count = inv_count + (mid - i);
```

```
        }
    }

    /* Copy the remaining elements of left
    subarray (if there are any) to temp*/
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    /* Copy the remaining elements of right
    subarray (if there are any) to temp*/
    while (j <= right)
        temp[k++] = arr[j++];

    // Copy back the merged elements
    // to original array
    for (i = left; i <= right; i++)
        arr[i] = temp[i];

    return inv_count;
}

/* An auxiliary recursive function
that sorts the input array and
returns the number of inversions
in the array. */
static int _mergeSort(int []arr, int []temp,
                        int left,int right)
{
    int mid, inv_count = 0;
    if (right > left)
    {
        /* Divide the array into two parts and call
        _mergeSortAndCountInv() for each of
        the parts */
        mid = (right + left) / 2;

        // Inversion count will be sum of inversions in
        // left-part, right-part and number of inversions
        // in merging
        inv_count = _mergeSort(arr, temp, left, mid);
        inv_count += _mergeSort(arr, temp, mid+1, right);

        /*Merge the two parts*/
        inv_count += merge(arr, temp, left, mid+1, right);
    }

    return inv_count;
}
```

```
    // This function sorts the input array and
    // returns the number of inversions in the
    // array
    static int countPairs(int []arr, int n)
    {
        // Modify the array so that problem reduces to
        // count inversion problem.
        for (int i = 0; i < n; i++)
            arr[i] = i * arr[i];

        // Count inversions using same logic as
        // below post
        // https://www.geeksforgeeks.org/counting-inversions/
        int []temp = new int [n];
        return _mergeSort(arr, temp, 0, n - 1);
    }

    // Driver code

    public static void Main ()
    {
        int []arr = {5, 0, 10, 2, 4, 1, 6};
        int n = arr.Length;
        Console.WriteLine( "Count of Pairs : "
                        + countPairs(arr, n));

    }
}

// This code is contributed by anuj_67.
```

Output:

```
Count of Pairs : 5
```

Time Complexity: O(n log n)

Improved By : Sam007, jit_t, vt_m

Source

https://www.geeksforgeeks.org/count-pairs-array-hold-iarri-jarrj/

Chapter 38

Count pairs in array whose sum is divisible by 4

Count pairs in array whose sum is divisible by 4 - GeeksforGeeks

Given a array if 'n' positive integers. Count number of pairs of integers in the array that have the sum divisible by 4.
Examples :

```
Input: {2, 2, 1, 7, 5}
Output: 3

Explanation
Only three pairs are possible whose sum
is divisible by '4' i.e., (2, 2),
(1, 7) and (7, 5)

Input: {2, 2, 3, 5, 6}
Output: 4
```

Naive approach is to iterate through every pair of array bu using two nested for loops and count those pairs whose sum is divisible by '4'. Time complexity of this approach is $O(n^2)$.

Efficient approach is to use Hashing technique. There are only three condition that can arise whose sum is divisible by '4' i.e,

1. If both are divisible by 4.
2. If one of them is equal to **1 modulo 4** and other is **3 modulo 4**. For instance, (1, 3), (5, 7), (5, 11).
3. If both of them is equal to **2 modulo 4** i.e., (2, 2), (2, 6), (6, 10)

```
Thus answer =>
```

C++

```
 // C++ Program to count pairs
// whose sum divisible by '4'
#include <bits/stdc++.h>
using namespace std;

// Program to count pairs whose sum divisible
// by '4'
int count4Divisibiles(int arr[], int n)
{
    // Create a frequency array to count
    // occurrences of all remainders when
    // divided by 4
    int freq[4] = {0, 0, 0, 0};

    // Count occurrences of all remainders
    for (int i = 0; i < n; i++)
        ++freq[arr[i] % 4];

    // If both pairs are divisible by '4'
    int ans = freq[0] * (freq[0] - 1) / 2;

    // If both pairs are 2 modulo 4
    ans += freq[2] * (freq[2] - 1) / 2;

    // If one of them is equal
    // to 1 modulo 4 and the
    // other is equal to 3
    // modulo 4
    ans += freq[1] * freq[3];

    return ans;
}

// Driver code
int main()
{

    int arr[] = { 2, 2, 1, 7, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    cout << count4Divisibiles(arr, n);

    return 0;
}
```

Java

```
 // Java program to count pairs
// whose sum divisible by '4'
import java.util.*;

class Count{
    public static int count4Divisibiles(int arr[] ,
                                            int n )
    {
        // Create a frequency array to count
        // occurrences of all remainders when
        // divided by 4
        int freq[] = {0, 0, 0, 0};
        int i = 0;
        int ans;

        // Count occurrences of all remainders
        for (i = 0; i < n; i++)
                ++freq[arr[i] % 4];

        //If both pairs are divisible by '4'
        ans = freq[0] * (freq[0] - 1) / 2;

        // If both pairs are 2 modulo 4
        ans += freq[2] * (freq[2] - 1) / 2;

        // If one of them is equal
        // to 1 modulo 4 and the
        // other is equal to 3
        // modulo 4
        ans += freq[1] * freq[3];

        return (ans);
    }
    public static void main(String[] args)
    {
        int arr[] = {2, 2, 1, 7, 5};
        int n = 5;
        System.out.print(count4Divisibiles(arr, n));
    }
}
```

```
// This code is contributed by rishabh_jain
```

Python3

```
 # Python3 code to count pairs whose
# sum is divisible by '4'

# Function to count pairs whose
# sum is divisible by '4'
def count4Divisibiles( arr , n ):

    # Create a frequency array to count
    # occurrences of all remainders when
    # divided by 4
    freq = [0, 0, 0, 0]

    # Count occurrences of all remainders
    for i in range(n):
        freq[arr[i] % 4]+=1

    #If both pairs are divisible by '4'
    ans = freq[0] * (freq[0] - 1) / 2

    # If both pairs are 2 modulo 4
    ans += freq[2] * (freq[2] - 1) / 2

    # If one of them is equal
    # to 1 modulo 4 and the
    # other is equal to 3
    # modulo 4
    ans += freq[1] * freq[3]

    return int(ans)

# Driver code
arr = [2, 2, 1, 7, 5]
n = len(arr)
print(count4Divisibiles(arr, n))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# program to count pairs
// whose sum divisible by '4'
using System;
```

```
class Count{
    public static int count4Divisibiles(int []arr ,
                                        int n )
    {
        // Create a frequency array to count
        // occurrences of all remainders when
        // divided by 4
        int []freq = {0, 0, 0, 0};
        int i = 0;
        int ans;

        // Count occurrences of all remainders
        for (i = 0; i < n; i++)
            ++freq[arr[i] % 4];

        //If both pairs are divisible by '4'
        ans = freq[0] * (freq[0] - 1) / 2;

        // If both pairs are 2 modulo 4
        ans += freq[2] * (freq[2] - 1) / 2;

        // If one of them is equal
        // to 1 modulo 4 and the
        // other is equal to 3
        // modulo 4
        ans += freq[1] * freq[3];

        return (ans);
    }

    // Driver code
    public static void Main()
    {
        int []arr = {2, 2, 1, 7, 5};
        int n = 5;
        Console.WriteLine(count4Divisibiles(arr, n));
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP Program to count pairs
// whose sum divisible by '4'
```

```
// Program to count pairs whose
// sum divisible by '4'
function count4Divisibiles($arr, $n)
{
    // Create a frequency array to
    // count occurrences of all
    // remainders when divided by 4
    $freq = array(0, 0, 0, 0);

    // Count occurrences
    // of all remainders
    for ( $i = 0; $i < $n; $i++)
        ++$freq[$arr[$i] % 4];

    // If both pairs are
    // divisible by '4'
    $ans = $freq[0] *
          ($freq[0] - 1) / 2;

    // If both pairs are
    // 2 modulo 4
    $ans += $freq[2] *
           ($freq[2] - 1) / 2;

    // If one of them is equal
    // to 1 modulo 4 and the
    // other is equal to 3
    // modulo 4
    $ans += $freq[1] * $freq[3];

    return $ans;
}

// Driver code
$arr = array(2, 2, 1, 7, 5);
$n = sizeof($arr) ;

echo count4Divisibiles($arr, $n);

// This code is contributed by ajit
?>
```

Output :

```
 3
```

Time complexity: O(n)

Auxiliary space: O(1)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/count-pairs-array-whose-sum-divisible-4/

Chapter 39

Count palindromic characteristics of a String

Count palindromic characteristics of a String - GeeksforGeeks

Given a string s of length n, count the number of substrings having different types of palindromic characteristics.
Palindromic Characteristic is the number of k-palindromes in a string where k lies in range [0, n).
A string is **1-palindrome**(or simply palindrome) if and only if it reads same the backward as it reads forward.
A string is **k-palindrome** (k > 1) if and only if :
1. Its left half is equal to its right half.
2. Its left half and right half should be non-empty (k – 1)-palindrome. The left half of a string of length 't' is its prefix of length $\lfloor t/2 \rfloor$, and right half is the suffix of the same length.
Note : Each substring is counted as many times as it appears in the string. For example, in the string "aaa" the substring "a" appears 3 times.

Examples :

```
Input : abba
Output : 6 1 0 0
Explanation :
"6" 1-palindromes = "a", "b", "b", "a", "bb", "abba".
"1" 2-palindrome = "bb". Because "b" is 1-palindrome
and "bb" has both left and right parts equal.
"0" 3-palindrome and 4-palindrome.

Input : abacaba
Output : 12 4 1 0 0 0 0
Explanation :
"12" 1-palindromes = "a", "b", "a", "c", "a", "b",
```

```
"a", "aba", "aca", "aba", "bacab", "abacaba".
"4" 2-palindromes = "aba", "aca", "aba", "abacaba".
Because "a" and "aba" are 1-palindromes.
NOTE :  "bacab" is not 2-palindrome as "ba"
is not 1-palindrome. "1" 3-palindrome = "abacaba".
Because is "aba" is 2-palindrome. "0" other
k-pallindromes where 4 < = k < = 7.
```

Approach : Take a string s and say it is a 1-palindrome and checking its left half also turns out to be a 1-palindrome then, obviously its right part will always be equal to left part (as the string is also a 1-palindrome) making the original string to be 2-palindrome. Now, similarly, checking the left half of the string, it also turns out to be a 1-palindrome then it will make left half to be 2-palindrome and hence, making original string to be 3-palindrome and so on checking it till only 1 alphabet is left or the part is not a 1-palindrome.
Lets take string = "abacaba". As known "abacaba" is 1-palindrome. So, when checking for its left half "aba", which is also a 1-palindrome, it makes "abacaba" a 2-palindrome. Then, check for "aba"'s left half "a" which is also a 1-palindrome. So it makes "aba" a 2-palindrome and "aba" makes "abacaba" a 3-palindrome. So in other words, if a string is a k-palindrome then it is also a (k-1)-palindrome, (k-2)-palindrome and so on till 1-palindrome. Code for the same is given below which firsts check each and every substring if it is a palindrome or not and then counts the number of k-palindromic substrings recursively in logarithmic time.

Below is the implementation of above approach :

C++

```
 // A C++ program which counts different
// palindromic characteristics of a string.
#include <bits/stdc++.h>
using namespace std;

const int MAX_STR_LEN = 1000;

bool P[MAX_STR_LEN][MAX_STR_LEN];
int Kpal[MAX_STR_LEN];

// A C++ function which checks whether a
// substring str[i..j] of a given string
// is a palindrome or not.
void checkSubStrPal(string str, int n)
{

    // P[i][j] = true if substring str
    // [i..j] is palindrome, else false
    memset(P, false, sizeof(P));

    // palindrome of single length
    for (int i = 0; i < n; i++)
```

```
        P[i][i] = true;

    // palindrome of length 2
    for (int i = 0; i < n - 1; i++)
        if (str[i] == str[i + 1])
            P[i][i + 1] = true;

    // Palindromes of length more then 2.
    // This loop is similar to Matrix Chain
    // Multiplication. We start with a gap of
    // length 2 and fill P table in a way that
    // gap between starting and ending indexes
    // increases one by one by outer loop.
    for (int gap = 2; gap < n; gap++)
    {

        // Pick starting point for current gap
        for (int i = 0; i < n - gap; i++)
        {

            // Set ending point
            int j = gap + i;

            // If current string is palindrome
            if (str[i] == str[j] && P[i + 1][j - 1])
                P[i][j] = true;
        }
    }
}

// A C++ function which recursively
// counts if a string str [i..j] is
// a k-palindromic string or not.
void countKPalindromes(int i, int j, int k)
{
    // terminating condition for a
    // string which is a k-palindrome.
    if (i == j)
    {
        Kpal[k]++;
        return;
    }

    // terminating condition for a
    // string which is not a k-palindrome.
    if (P[i][j] == false)
        return;
```

```
    // increases the counter for the
    // string if it is a k-palindrome.
    Kpal[k]++;

    // mid is middle pointer of
    // the string str [i...j].
    int mid = (i + j) / 2;

    // if length of string which is
    // (j - i + 1) is odd than we have
    // to subtract one from mid.
    // else if even then no change.
    if ((j - i + 1) % 2 == 1)
        mid--;

    // if the string is k-palindrome
    // then we check if it is a
    // (k+1) - palindrome or not by
    // just sending any of one half of
    // the string to the Count_k_Palindrome
    // function.
    countKPalindromes(i, mid, k + 1);
}

void printKPalindromes(string s)
{

    // Count of k-palindromes is equal
    // to zero initially.
    memset(Kpal, 0, sizeof(Kpal));

    // Finding all palindromic
    // substrings of given string
    int n = s.length();
    checkSubStrPal(s, n);

    // counting k-palindromes for each and
    // every substring of given string. .
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n - i; j++)
            countKPalindromes(j, j + i, 1);

    // Output the number of K-palindromic
    // substrings of a given string.
    for (int i = 1; i <= n; i++)
        cout << Kpal[i] << " ";
    cout << "\n";
}
```

```
// Driver code
int main()
{
    string s = "abacaba";
    printKPalindromes(s);
    return 0;
}
```

Java

```
 // Java program which counts
// different palindromic
// characteristics of a string.
import java.io.*;

class GFG
{
    static int MAX_STR_LEN = 1000;

    static boolean P[][] =
           new boolean[MAX_STR_LEN][MAX_STR_LEN];
    static int []Kpal =
           new int[MAX_STR_LEN];

    // function which checks
    // whether a substring
    // str[i..j] of a given
    // string is a palindrome or not.
    static void checkSubStrPal(String str,
                               int n)
    {

        // P[i,j] = true if substring
        // str [i..j] is palindrome,
        // else false
        for (int i = 0; i < MAX_STR_LEN; i++)
        {
            for (int j = 0; j < MAX_STR_LEN; j++)
                P[i][j] = false;
            Kpal[i] = 0;
        }

        // palindrome of
        // single length
        for (int i = 0; i < n; i++)
            P[i][i] = true;
```

```
    // palindrome of
    // length 2
    for (int i = 0; i < n - 1; i++)
        if (str.charAt(i) == str.charAt(i + 1))
            P[i][i + 1] = true;

    // Palindromes of length
    // more then 2. This loop
    // is similar to Matrix
    // Chain Multiplication.
    // We start with a gap of
    // length 2 and fill P table
    // in a way that gap between
    // starting and ending indexes
    // increases one by one by
    // outer loop.
    for (int gap = 2; gap < n; gap++)
    {

        // Pick starting point
        // for current gap
        for (int i = 0; i < n - gap; i++)
        {

            // Set ending point
            int j = gap + i;

            // If current string
            // is palindrome
            if (str.charAt(i) == str.charAt(j) &&
                                 P[i + 1][j - 1])
                P[i][j] = true;
        }
    }
}

// A function which recursively
// counts if a string str [i..j] is
// a k-palindromic string or not.
static void countKPalindromes(int i, int j,
                              int k)
{
    // terminating condition
    // for a string which is
    // a k-palindrome.
    if (i == j)
    {
        Kpal[k]++;
```

```
            return;
        }

        // terminating condition for
        // a string which is not a
        // k-palindrome.
        if (P[i][j] == false)
            return;

        // increases the counter
        // for the string if it
        // is a k-palindrome.
        Kpal[k]++;

        // mid is middle pointer of
        // the string str [i...j].
        int mid = (i + j) / 2;

        // if length of string which
        // is (j - i + 1) is odd than
        // we have to subtract one
        // from mid else if even then
        // no change.
        if ((j - i + 1) % 2 == 1)
            mid--;

        // if the string is k-palindrome
        // then we check if it is a
        // (k+1) - palindrome or not
        // by just sending any of one
        // half of the string to the
        // Count_k_Palindrome function.
        countKPalindromes(i, mid, k + 1);
    }

    static void printKPalindromes(String s)
    {
        // Finding all palindromic
        // substrings of given string
        int n = s.length();
        checkSubStrPal(s, n);

        // counting k-palindromes for
        // each and every substring
        // of given string. .
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n - i; j++)
                countKPalindromes(j, j + i, 1);
```

```
        // Output the number of
        // K-palindromic substrings
        // of a given string.
        for (int i = 1; i <= n; i++)
            System.out.print(Kpal[i] + " ");
        System.out.println();
    }

    // Driver code
    public static void main(String args[])
    {
        String s = "abacaba";
        printKPalindromes(s);
    }
}

// This code is contributd by
// Manish Shaw(manishshaw1)
```

Python3

```
 # Python program which counts
# different palindromic
# characteristics of a string.
MAX_STR_LEN = 1000;

P = [[0 for x in range(MAX_STR_LEN)]
        for y in range(MAX_STR_LEN)] ;

for i in range(0, MAX_STR_LEN) :
    for j in range(0, MAX_STR_LEN) :
        P[i][j] = False;

Kpal = [0] * MAX_STR_LEN;

# def which checks
# whether a substr[i..j]
# of a given is a
# palindrome or not.
def checkSubStrPal(str, n) :

    global P, Kpal, MAX_STR_LEN;

    # P[i,j] = True if substr
    # [i..j] is palindrome,
    # else False
    for i in range(0, MAX_STR_LEN) :
```

```
        for j in range(0, MAX_STR_LEN) :
            P[i][j] = False;
        Kpal[i] = 0;

    # palindrome of
    # single length
    for i in range(0, n) :
        P[i][i] = True;

    # palindrome of
    # length 2
    for i in range(0, n - 1) :
        if (str[i] == str[i + 1]) :
            P[i][i + 1] = True;

    # Palindromes of length more
    # then 2. This loop is similar
    # to Matrix Chain Multiplication.
    # We start with a gap of length
    # 2 and fill P table in a way
    # that gap between starting and
    # ending indexes increases one
    # by one by outer loop.
    for gap in range(2, n) :

        # Pick starting point
        # for current gap
        for i in range(0, n - gap) :

            # Set ending point
            j = gap + i;

            # If current string
            # is palindrome
            if (str[i] == str[j] and
                P[i + 1][j - 1]) :
                P[i][j] = True;

# A Python def which
# recursively counts if
# a str [i..j] is a
# k-palindromic or not.
def countKPalindromes(i, j, k) :

    global Kpal, P;

    # terminating condition
```

```
    # for a which is a
    # k-palindrome.
    if (i == j) :

        Kpal[k] = Kpal[k] + 1;
        return;

    # terminating condition
    # for a which is not a
    # k-palindrome.
    if (P[i][j] == False) :
        return;

    # increases the counter
    # for the if it is a
    # k-palindrome.
    Kpal[k] = Kpal[k] + 1;

    # mid is middle pointer
    # of the str [i...j].
    mid = int((i + j) / 2);

    # if length of which is
    # (j - i + 1) is odd than
    # we have to subtract one
    # from mid else if even
    # then no change.
    if ((j - i + 1) % 2 == 1) :
        mid = mid - 1;

    # if the is k-palindrome
    # then we check if it is a
    # (k+1) - palindrome or not
    # by just sending any of
    # one half of the to the
    # Count_k_Palindrome def.
    countKPalindromes(i, mid, k + 1);

def printKPalindromes(s) :

    global P, Kpal, MAX_STR_LEN;

    # Finding all palindromic
    # substrings of given string
    n = len(s);
    checkSubStrPal(s, n);

    # counting k-palindromes
```

```
    # for each and every sub
    # of given string. .
    for i in range(0, n) :
        for j in range(0, n - i) :
            countKPalindromes(j, j + i, 1);

    # Output the number of
    # K-palindromic substrings
    # of a given string.
    for i in range(1, n + 1) :
        print (Kpal[i], end=" ");
    print();


# Driver code
s = "abacaba";
printKPalindromes(s);

# This code is contributd by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# program which counts
// different palindromic
// characteristics of a string.
using System;

class GFG
{
    static int MAX_STR_LEN = 1000;

    static bool [,]P = new bool[MAX_STR_LEN,
                               MAX_STR_LEN];
    static int []Kpal = new int[MAX_STR_LEN];

    // function which checks whether
    // a substring str[i..j] of a
    // given string is a palindrome or not.
    static void checkSubStrPal(string str,
                               int n)
    {

        // P[i,j] = true if substring str
        // [i..j] is palindrome, else false
        for (int i = 0; i < MAX_STR_LEN; i++)
        {
            for (int j = 0; j < MAX_STR_LEN; j++)
```

```
            P[i, j] = false;
        Kpal[i] = 0;
    }

    // palindrome of single length
    for (int i = 0; i < n; i++)
        P[i, i] = true;

    // palindrome of length 2
    for (int i = 0; i < n - 1; i++)
        if (str[i] == str[i + 1])
            P[i, i + 1] = true;

    // Palindromes of length more
    // then 2. This loop is similar
    // to Matrix Chain Multiplication.
    // We start with a gap of length 2
    // and fill P table in a way that
    // gap between starting and ending
    // indexes increases one by one by
    // outer loop.
    for (int gap = 2; gap < n; gap++)
    {

        // Pick starting point
        // for current gap
        for (int i = 0; i < n - gap; i++)
        {

            // Set ending point
            int j = gap + i;

            // If current string
            // is palindrome
            if (str[i] == str[j] &&
                    P[i + 1, j - 1])
                P[i, j] = true;
        }
    }
}

// A C++ function which recursively
// counts if a string str [i..j] is
// a k-palindromic string or not.
static void countKPalindromes(int i, int j,
                              int k)
{
    // terminating condition for a
```

```
    // string which is a k-palindrome.
    if (i == j)
    {
        Kpal[k]++;
        return;
    }

    // terminating condition for
    // a string which is not a
    // k-palindrome.
    if (P[i, j] == false)
        return;

    // increases the counter for the
    // string if it is a k-palindrome.
    Kpal[k]++;

    // mid is middle pointer of
    // the string str [i...j].
    int mid = (i + j) / 2;

    // if length of string which is
    // (j - i + 1) is odd than we have
    // to subtract one from mid.
    // else if even then no change.
    if ((j - i + 1) % 2 == 1)
        mid--;

    // if the string is k-palindrome
    // then we check if it is a
    // (k+1) - palindrome or not
    // by just sending any of one
    // half of the string to the
    // Count_k_Palindrome function.
    countKPalindromes(i, mid, k + 1);
}

static void printKPalindromes(string s)
{
    // Finding all palindromic
    // substrings of given string
    int n = s.Length;
    checkSubStrPal(s, n);

    // counting k-palindromes for each and
    // every substring of given string. .
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n - i; j++)
```

```
                countKPalindromes(j, j + i, 1);

        // Output the number of K-palindromic
        // substrings of a given string.
        for (int i = 1; i <= n; i++)
            Console.Write(Kpal[i] + " ");
        Console.WriteLine();
    }

    // Driver code
    static void Main()
    {
        string s = "abacaba";
        printKPalindromes(s);
    }
}

// This code is contributd by
// Manish Shaw(manishshaw1)
```

PHP

```
 <?php
// PHP program which counts
// different palindromic
// characteristics of a string.
$MAX_STR_LEN = 1000;

$P = array(array());
$Kpal = array_fill(0, $MAX_STR_LEN, 0);

for($i = 0; $i < $MAX_STR_LEN; $i++)
{
    for($j = 0; $j < $MAX_STR_LEN; $j++)
        $P[$i][$j] = false;
}

// function which checks
// whether a substr[i..j]
// of a given is a
// palindrome or not.
function checkSubStrPal($str,
                        $n)
{
    global $P, $Kpal,
           $MAX_STR_LEN;

    // P[i,j] = true if substr
```

```
    // [i..j] is palindrome, else false
    for ($i = 0;
         $i < $MAX_STR_LEN; $i++)
    {
        for ($j = 0;
             $j < $MAX_STR_LEN; $j++)
            $P[$i][$j] = false;
        $Kpal[$i] = 0;
    }

    // palindrome of
    // single length
    for ($i = 0; $i < $n; $i++)
        $P[$i][$i] = true;

    // palindrome of
    // length 2
    for ($i = 0; $i < $n - 1; $i++)
        if ($str[$i] == $str[$i + 1])
            $P[$i][$i + 1] = true;

    // Palindromes of length more
    // then 2. This loop is similar
    // to Matrix Chain Multiplication.
    // We start with a gap of length
    // 2 and fill P table in a way
    // that gap between starting and
    // ending indexes increases one
    // by one by outer loop.
    for ($gap = 2; $gap < $n; $gap++)
    {
        // Pick starting point
        // for current gap
        for ($i = 0;
             $i < $n - $gap; $i++)
        {
            // Set ending point
            $j = $gap + $i;

            // If current string
            // is palindrome
            if ($str[$i] == $str[$j] &&
                    $P[$i + 1][$j - 1])
                $P[$i][$j] = true;
        }
    }
}
```

```
// A PHP function which
// recursively counts if
// a str [i..j] is a
// k-palindromic or not.
function countKPalindromes($i, $j, $k)
{
    global $Kpal, $P;

    // terminating condition for a
    // which is a k-palindrome.
    if ($i == $j)
    {
        $Kpal[$k]++;
        return;
    }

    // terminating condition
    // for a which is not a
    // k-palindrome.
    if ($P[$i][$j] == false)
        return;

    // increases the counter
    // for the if it is a
    // k-palindrome.
    $Kpal[$k]++;

    // mid is middle pointer
    // of the str [i...j].
    $mid = ($i + $j) / 2;

    // if length of which is
    // (j - i + 1) is odd than
    // we have to subtract one
    // from mid else if even
    // then no change.
    if (($j - $i + 1) % 2 == 1)
        $mid--;

    // if the is k-palindrome
    // then we check if it is a
    // (k+1) - palindrome or not
    // by just sending any of
    // one half of the to the
    // Count_k_Palindrome function.
    countKPalindromes($i, $mid,
                      $k + 1);
}
```

```
function printKPalindromes($s)
{
    global $P, $Kpal,
           $MAX_STR_LEN;

    // Finding all palindromic
    // substrings of given string
    $n = strlen($s);
    checkSubStrPal($s, $n);

    // counting k-palindromes
    // for each and every sub
    // of given string. .
    for ($i = 0; $i < $n; $i++)
        for ($j = 0; $j < $n - $i; $j++)
            countKPalindromes($j, $j +
                              $i, 1);

    // Output the number of
    // K-palindromic substrings
    // of a given string.
    for ($i = 1; $i <= $n; $i++)
        echo ($Kpal[$i] . " ");
    echo("\n");
}

// Driver code
$s = "abacaba";
printKPalindromes($s);

// This code is contributd by
// Manish Shaw(manishshaw1)
?>
```

Output :

```
12 4 1 0 0 0 0
```

Time Complexity : O(

Auxiliary Space : O(

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/count-palindromic-characteristics-string/

Chapter 40

Count quadruples from four sorted arrays whose sum is equal to a given value x

Count quadruples from four sorted arrays whose sum is equal to a given value x - GeeksforGeeks

Given four sorted arrays each of size **n** of distinct elements. Given a value **x**. The problem is to count all **quadruples**(group of four numbers) from all the four arrays whose sum is equal to **x**.

Note: The quadruple has an element from each of the four arrays.

Examples:

```
Input : arr1 = {1, 4, 5, 6},
        arr2 = {2, 3, 7, 8},
        arr3 = {1, 4, 6, 10},
        arr4 = {2, 4, 7, 8}
        n = 4, x = 30

Output : 4
The quadruples are:
(4, 8, 10, 8), (5, 7, 10, 8),
(5, 8, 10, 7), (6, 7, 10, 7)

Input : For the same above given fours arrays
        x = 25
Output : 14
```

Method 1 (Naive Approach): Using four nested loops generate all quadruples and check whether elements in the quadruple sum up to **x** or not.

C++

```
// C++ implementation to count quadruples from four sorted arrays
// whose sum is equal to a given value x
#include <bits/stdc++.h>

using namespace std;

// function to count all quadruples from
// four sorted arrays whose sum is equal
// to a given value x
int countQuadruples(int arr1[], int arr2[],
                    int arr3[], int arr4[], int n, int x)
{
    int count = 0;

    // generate all possible quadruples from
    // the four sorted arrays
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                for (int l = 0; l < n; l++)
                    // check whether elements of
                    // quadruple sum up to x or not
                    if ((arr1[i] + arr2[j] + arr3[k] + arr4[l]) == x)
                        count++;

    // required count of quadruples
    return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };

    int n = sizeof(arr1) / sizeof(arr1[0]);
    int x = 30;
    cout << "Count = "
         << countQuadruples(arr1, arr2, arr3,
                            arr4, n, x);
    return 0;
}
```

Python3

```
 # A Python implementation to count
# quadruples from four sorted arrays
# whose sum is equal to a given value x

# function to count all quadruples
# from four sorted arrays whose sum
# is equal to a given value x
def countQuuadruples(arr1, arr2,
                     arr3, arr4, n, x):
    count = 0

    # generate all possible
    # quadruples from the four
    # sorted arrays
    for i in range(n):
        for j in range(n):
            for k in range(n):
                for l in range(n):

                    # check whether elements of
                    # quadruple sum up to x or not
                    if (arr1[i] + arr2[j] +
                        arr3[k] + arr4[l] == x):
                        count += 1

    # required count of quadruples
    return count

# Driver Code
arr1 = [1, 4, 5, 6]
arr2 = [2, 3, 7, 8]
arr3 = [1, 4, 6, 10]
arr4 = [2, 4, 7, 8 ]
n = len(arr1)
x = 30
print("Count = ", countQuuadruples(arr1, arr2,
                                   arr3, arr4, n, x))

# This code is contributed
# by Shrikant13
```

Output:

```
Count = 4
```

Time Complexity: O(n^4)
Auxiliary Space: O(1)

Method 2 (Binary Search): Generate all triplets from the 1st three arrays. For each triplet so generated, find the sum of elements in the triplet. Let it be **T**. Now, search the value **(x − T)** in the 4th array. If value found in the 4th array, then increment **count**. This process is repeated for all the triplets generated from the 1st three arrays.

```
 // C++ implementation to count quadruples from
// four sorted arrays whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

// find the 'value' in the given array 'arr[]'
// binary search technique is applied
bool isPresent(int arr[], int low, int high, int value)
{
    while (low <= high) {
        int mid = (low + high) / 2;

        // 'value' found
        if (arr[mid] == value)
            return true;
        else if (arr[mid] > value)
            high = mid - 1;
        else
            low = mid + 1;
    }

    // 'value' not found
    return false;
}

// function to count all quadruples from four
// sorted arrays whose sum is equal to a given value x
int countQuadruples(int arr1[], int arr2[], int arr3[],
                    int arr4[], int n, int x)
{
    int count = 0;

    // generate all triplets from the 1st three arrays
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++) {

                // calculate the sum of elements in
                // the triplet so generated
```

```
                int T = arr1[i] + arr2[j] + arr3[k];

                // check if 'x-T' is present in 4th
                // array or not
                if (isPresent(arr4, 0, n, x - T))

                    // increment count
                    count++;
            }

    // required count of quadruples
    return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };

    int n = sizeof(arr1) / sizeof(arr1[0]);
    int x = 30;
    cout << "Count = "
         << countQuadruples(arr1, arr2, arr3, arr4, n, x);
    return 0;
}
```

Output:

```
Count = 4
```

Time Complexity: O(n^3logn)
Auxiliary Space: O(1)

Method 3 (Use of two pointers): Generate all pairs from the 1st two arrays. For each pair so generated, find the sum of elements in the pair. Let it be **p_sum**. For each **p_sum**, count pairs from the 3rd and 4th sorted array with sum equal to **(x − p_sum)**. Accumulate these count in the **total_count** of quadruples.

```
 // C++ implementation to count quadruples from
// four sorted arrays whose sum is equal to a
// given value x
#include <bits/stdc++.h>
```

```
using namespace std;

// count pairs from the two sorted array whose sum
// is equal to the given 'value'
int countPairs(int arr1[], int arr2[], int n, int value)
{
    int count = 0;
    int l = 0, r = n - 1;

    // traverse 'arr1[]' from left to right
    // traverse 'arr2[]' from right to left
    while (l < n & amp; &r >= 0) {
        int sum = arr1[l] + arr2[r];

        // if the 'sum' is equal to 'value', then
        // increment 'l', decrement 'r' and
        // increment 'count'
        if (sum == value) {
            l++, r--;
            count++;
        }

        // if the 'sum' is greater than 'value', then
        // decrement r
        else if (sum > value)
            r--;

        // else increment l
        else
            l++;
    }

    // required count of pairs
    return count;
}

// function to count all quadruples from four sorted arrays
// whose sum is equal to a given value x
int countQuadruples(int arr1[], int arr2[], int arr3[],
                    int arr4[], int n, int x)
{
    int count = 0;

    // generate all pairs from arr1[] and arr2[]
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            // calculate the sum of elements in
            // the pair so generated
```

```
            int p_sum = arr1[i] + arr2[j];

            // count pairs in the 3rd and 4th array
            // having value 'x-p_sum' and then
            // accumulate it to 'count'
            count += countPairs(arr3, arr4, n, x - p_sum);
        }

    // required count of quadruples
    return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };

    int n = sizeof(arr1) / sizeof(arr1[0]);
    int x = 30;
    cout << "Count = "
         << countQuadruples(arr1, arr2, arr3,
                            arr4, n, x);
    return 0;
}
```

Output:

```
Count = 4
```

Time Complexity: O(n^3)
Auxiliary Space: O(1)

Method 4 Efficient Approach(Hashing): Create a hash table where **(key, value)** tuples are represented as **(sum, frequency)** tuples. Here the sum are obtained from the pairs of 1st and 2nd array and their frequency count is maintained in the hash table. Hash table is implemented using unordered_map in C++. Now, generate all pairs from the 3rd and 4th array. For each pair so generated, find the sum of elements in the pair. Let it be **p_sum**. For each **p_sum**, check whether **(x − p_sum)** exists in the hash table or not. If it exists, then add the frequency of **(x − p_sum)** to the **count** of quadruples.

```
 // C++ implementation to count quadruples from
// four sorted arrays whose sum is equal to a
// given value x
```

```
#include <bits/stdc++.h>

using namespace std;

// function to count all quadruples from four sorted
// arrays whose sum is equal to a given value x
int countQuadruples(int arr1[], int arr2[], int arr3[],
                    int arr4[], int n, int x)
{
    int count = 0;

    // unordered_map 'um' implemented as hash table
    // for <sum, frequency> tuples
    unordered_map<int, int> um;

    // count frequency of each sum obtained from the
    // pairs of arr1[] and arr2[] and store them in 'um'
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            um[arr1[i] + arr2[j]]++;

    // generate pair from arr3[] and arr4[]
    for (int k = 0; k < n; k++)
        for (int l = 0; l < n; l++) {

            // calculate the sum of elements in
            // the pair so generated
            int p_sum = arr3[k] + arr4[l];

            // if 'x-p_sum' is present in 'um' then
            // add frequency of 'x-p_sum' to 'count'
            if (um.find(x - p_sum) != um.end())
                count += um[x - p_sum];
        }

    // required count of quadruples
    return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };
```

```
    int n = sizeof(arr1) / sizeof(arr1[0]);
    int x = 30;
    cout << "Count = "
         << countQuadruples(arr1, arr2, arr3, arr4, n, x);
    return 0;
}
```

Output:

```
Count = 4
```

Time Complexity: $O(n^2)$
Auxiliary Space: $O(n^2)$

Improved By : shrikanth13

Source

https://www.geeksforgeeks.org/count-quadruples-four-sorted-arrays-whose-sum-equal-given-value-x/

Chapter 41

Count ways of choosing a pair with maximum difference

Count ways of choosing a pair with maximum difference - GeeksforGeeks

Given an array of n integers, we need to find the no. of ways of choosing pairs with maximum difference.

Examples:

```
Input : a[] = {3, 2, 1, 1, 3}
Output : 4
Explanation:- Here, the maximum difference
you can find is 2 which is from (1, 3).
No. of ways of choosing it:
 1) Choosing the first and third elements,
 2) Choosing the first and fourth elements,
 3) Choosing the third and fifth elements,
 4) Choosing the fourth and fifth elements.
Hence ans is 4.

Input : a[] = {2, 4, 1, 1}
Output : 2
Explanation:- Here, the maximum difference
is 3 from (1, 4). No. of ways choosing it:
1) Choosing the second and third elements,
2) Choosing the second and fourth elements.
   Hence ans is 2.
```

Naive Approach : A Simple solution is to find the minimum element and maximum element to find the maximum difference. Then we can find the no. of ways of choosing a pair by running two loops. In the inner loop, check if the two elements(one in outer loop

and other in inner loop) are making maximum difference, if yes increase the count.at last output the count.
Time Complexity: O(n^2)
Auxiliary Space: O(1)

Efficient approach:
An efficient approach will be:

- Case I (if all the elements are equal): The ans is no. of ways of choosing 2 elements from a set of n elements $^nC_2$ which is n(n-1)/2.
- Case II (If all the elements are not equal) : The answer is product of count of no. of minimum elements(c1) and count of no. of maximum elements(c2), i.e., c1*c2

C++

```
 // CPP Code to find no. of Ways of choosing
// a pair with maximum difference
#include <bits/stdc++.h>
using namespace std;

int countPairs(int a[], int n)
{
    // To find minimum and maximum of
    // the array
    int mn = INT_MAX;
    int mx = INT_MIN;
    for (int i = 0; i < n; i++) {
        mn = min(mn, a[i]);
        mx = max(mx, a[i]);
    }

    // to find the count of minimum and
    // maximum elements
    int c1 = 0;
    int c2 = 0; // Count variables
    for (int i = 0; i < n; i++) {
        if (a[i] == mn)
            c1++;
        if (a[i] == mx)
            c2++;
    }

    // condition for all elements equal
    if (mn == mx)
        return n * (n - 1) / 2;
    else
        return c1 * c2;
```

```
}

// Driver code
int main()
{
    int a[] = { 3, 2, 1, 1, 3 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << countPairs(a, n);
    return 0;
}
```

Java

```
 // Java Code to find no. of Ways of choosing
// a pair with maximum difference
import java.util.*;

class GFG {

    static int countPairs(int a[], int n)
    {

        // To find minimum and maximum of
        // the array
        int mn = Integer.MAX_VALUE;
        int mx = Integer.MIN_VALUE;
        for (int i = 0; i < n; i++) {
            mn = Math.min(mn, a[i]);
            mx = Math.max(mx, a[i]);
        }

        // to find the count of minimum and
        // maximum elements
        int c1 = 0;
        int c2 = 0; // Count variables
        for (int i = 0; i < n; i++) {
            if (a[i] == mn)
                c1++;
            if (a[i] == mx)
                c2++;
        }

        // condition for all elements equal
        if (mn == mx)
            return n * (n - 1) / 2;
        else
            return c1 * c2;
    }
```

```
    // Driver code
    public static void main(String[] args)
    {
        int a[] = { 3, 2, 1, 1, 3 };
        int n = a.length;
        System.out.print(countPairs(a, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python Code to find no.
# of Ways of choosing
# a pair with maximum difference

def countPairs(a, n):

    # To find minimum and maximum of
    # the array
    mn = +2147483647
    mx = -2147483648
    for i in range(n):
        mn = min(mn, a[i])
        mx = max(mx, a[i])


    # to find the count of minimum and
    # maximum elements
    c1 = 0
    c2 = 0 # Count variables
    for i in range(n):
        if (a[i] == mn):
            c1+= 1
        if (a[i] == mx):
            c2+= 1


    # condition for all elements equal
    if (mn == mx):
        return  n*(n - 1) // 2
    else:
        return c1 * c2

# Driver code
```

```
a = [ 3, 2, 1, 1, 3]
n = len(a)

print(countPairs(a, n))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# Code to find no. of Ways of choosing
// a pair with maximum difference
using System;

class GFG {

    static int countPairs(int[] a, int n)
    {

        // To find minimum and maximum of
        // the array
        int mn = int.MaxValue;
        int mx = int.MinValue;
        for (int i = 0; i < n; i++) {
            mn = Math.Min(mn, a[i]);
            mx = Math.Max(mx, a[i]);
        }

        // to find the count of minimum and
        // maximum elements
        int c1 = 0;
        int c2 = 0; // Count variables
        for (int i = 0; i < n; i++) {
            if (a[i] == mn)
                c1++;
            if (a[i] == mx)
                c2++;
        }

        // condition for all elements equal
        if (mn == mx)
            return n * (n - 1) / 2;
        else
            return c1 * c2;
    }

    // Driver code
    public static void Main()
```

```
    {

        int[] a = { 3, 2, 1, 1, 3 };
        int n = a.Length;

        Console.WriteLine(countPairs(a, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Code to find no. of Ways of choosing
// a pair with maximum difference

function countPairs($a, $n)
{
    // To find minimum and maximum of
    // the array
    $mn = PHP_INT_MAX;
    $mx = PHP_INT_MIN;
    for ($i = 0; $i < $n; $i++) {
        $mn = min($mn, $a[$i]);
        $mx = max($mx, $a[$i]);
    }

    // to find the count of minimum and
    // maximum elements
    $c1 = 0;
    $c2 = 0; // Count variables
    for ($i = 0; $i < $n; $i++) {
        if ($a[$i] == $mn)
            $c1++;
        if ($a[$i] == $mx)
            $c2++;
    }

    // condition for all elements equal
    if ($mn == $mx)
        return $n * ($n - 1) / 2;
    else
        return $c1 * $c2;
}

// Driver code
```

```
    $a = array( 3, 2, 1, 1, 3 );
    $n = count($a);
    echo countPairs($a, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
  4
```

Time Complexity: Time complexity to find minimum and maximum is **O(n)** and Time Complexity to find count of minimum and maximum is **O(n)**
Overall Time complexity : O(n)
Auxiliary Space : O(1)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/count-ways-of-choosing-a-pair-with-maximum-difference/

Chapter 42

Count ways to form minimum product triplets

Count ways to form minimum product triplets - GeeksforGeeks

Given an array of positive integers. We need to find how many triples of indices (i, j, k) (i $<$ j $<$ k), such that a[i] * a[j] * a[k] is minimum possible.

```
Examples:
Input : 5
        1 3 2 3 4
Output : 2
The triplets are (1, 3, 2)
and (1, 2, 3)

Input : 5
        2 2 2 2 2
Output : 5
In this example we choose three 2s
out of five, and the number of ways
to choose them is 5C3.

Input : 6
        1 3 3 1 3 2
Output : 1
There is only one way (1, 1, 2).
```

Following cases arise in this problem.

1. All three minimum elements are same. For example {1, 1, 1, 1, 2, 3, 4}. The solution for such cases is ${}^{n}C_{3}$.

2. Two elements are same. For example {1, 2, 2, 2, 3} or {1, 1, 2, 2}. In this case, count of occurrences of first (or minimum element) cannot be more than 2. If minimuum element appears two times, then answer is count of second element (We get to choose only 1 from all occurrences of second element. If minimum element appears once, the count is ${}^{n}C_{2}$.
3. All three elements are distinct. For example {1, 2, 3, 3, 5}. In this case, answer is count of occurrences of third element (or ${}^{n}C_{1}$).

We first sort the array in increasing order. Then count the frequency of 3 element of 3rd element from starting. Let the frequency be 'count'. Following cases arise.

- If 3rd element is equal to the first element, no. of triples will be (count-2)*(count-1)*(count)/6, where count is the frequency of 3rd element.
- If 3rd element is equal to 2nd element, no. of triples will be (count-1)*(count)/2. Otherwise no. of triples will be value of count.

C++

```
 // CPP program to count number of ways we can
// form triplets with minimum product.
#include <bits/stdc++.h>
using namespace std;

// function to calculate number of triples
void noOfTriples(long long arr[], int n)
{
    // Sort the array
    sort(arr, arr + n);

    // Count occurrences of third element
    long long count = 0;
    for (long long i = 0; i < n; i++)
        if (arr[i] == arr[2])
            count++;

    // If all three elements are same (minimum
    // element appears at least 3 times). Answer
    // is nC3.
    if (arr[0] == arr[2])
        return (count - 2) * (count - 1) * (count) / 6;

    // If minimum element appears once.
    // Answer is nC2.
    else if (arr[1] == arr[2])
        return (count - 1) * (count) / 2;

    // Minimum two elements are distinct.
```

```
    // Answer is nC1.
    return count;
}

// Driver code
int main()
{
    long long arr[] = { 1, 3, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << noOfTriples(arr, n);
    return 0;
}
```

Java

```
 // Java program to count number of ways we can
// form triplets with minimum product.
import java.util.Arrays;

class GFG {

    // function to calculate number of triples
    static long noOfTriples(long arr[], int n)
    {

        // Sort the array
        Arrays.sort(arr);

        // Count occurrences of third element
        long count = 0;
        for (int i = 0; i < n; i++)
            if (arr[i] == arr[2])
                count++;

        // If all three elements are same (minimum
        // element appears at least 3 times). Answer
        // is nC3.
        if (arr[0] == arr[2])
            return (count - 2) * (count - 1) *
                                    (count) / 6;

        // If minimum element appears once.
        // Answer is nC2.
        else if (arr[1] == arr[2])
            return (count - 1) * (count) / 2;

        // Minimum two elements are distinct.
        // Answer is nC1.
```

```
        return count;
    }

    //driver code
    public static void main(String arg[])
    {

        long arr[] = { 1, 3, 3, 4 };
        int n = arr.length;

        System.out.print(noOfTriples(arr, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to count number
# of ways we can form triplets
# with minimum product.

# function to calculate number of triples
def noOfTriples (arr, n):

    # Sort the array
    arr.sort()

    # Count occurrences of third element
    count = 0
    for i in range(n):
        if arr[i] == arr[2]:
            count+=1

    # If all three elements are same
    # (minimum element appears at l
    # east 3 times). Answer is nC3.
    if arr[0] == arr[2]:
        return (count - 2) * (count - 1) * (count) / 6

    # If minimum element appears once.
    # Answer is nC2.
    elif arr[1] == arr[2]:
        return (count - 1) * (count) / 2

    # Minimum two elements are distinct.
    # Answer is nC1.
    return count
```

```
# Driver code
arr = [1, 3, 3, 4]
n = len(arr)
print (noOfTriples(arr, n))

# This code is contributed by "Abhishek Sharma 44"
```

C#

```
 // C# program to count number of ways
// we can form triplets with minimum product.
using System;

class GFG {

// function to calculate number of triples
static long noOfTriples(long []arr, int n)
{
    // Sort the array
    Array.Sort(arr);

    // Count occurrences of third element
    long count = 0;
    for (int i = 0; i < n; i++)
        if (arr[i] == arr[2])
            count++;

    // If all three elements are same (minimum
    // element appears at least 3 times). Answer
    // is nC3.
    if (arr[0] == arr[2])
        return (count - 2) * (count - 1) * (count) / 6;

    // If minimum element appears once.
    // Answer is nC2.
    else if (arr[1] == arr[2])
        return (count - 1) * (count) / 2;

    // Minimum two elements are distinct.
    // Answer is nC1.
    return count;
}

//driver code
public static void Main()
{
    long []arr = { 1, 3, 3, 4 };
```

```
    int n = arr.Length;
    Console.Write(noOfTriples(arr, n));
}
}

//This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// PHP program to count number of ways
// we can form triplets with minimum
// product.

// function to calculate number of
// triples
function noOfTriples( $arr, $n)
{
    // Sort the array
    sort($arr);

    // Count occurrences of third element
    $count = 0;
    for ( $i = 0; $i < $n; $i++)
        if ($arr[$i] == $arr[2])
            $count++;

    // If all three elements are same
    // (minimum element appears at least
    // 3 times). Answer is nC3.
    if ($arr[0] == $arr[2])
        return ($count - 2) * ($count - 1)
                            * ($count) / 6;

    // If minimum element appears once.
    // Answer is nC2.
    else if ($arr[1] == $arr[2])
        return ($count - 1) * ($count) / 2;

    // Minimum two elements are distinct.
    // Answer is nC1.
    return $count;
}

// Driver code
    $arr = array( 1, 3, 3, 4 );
    $n = count($arr);
    echo noOfTriples($arr, $n);
```

```
// This code is contributed by anuj_67.
?>
```

Output:

```
1
```

Time Complexity: O(n Log n)

The solution can be optimized by first finding minimum element and its frequency and if frequency is less than 3, then finding second minimum and its frequency. If overall frequency is less than 3, then finding third minimum and its frequency. Time complexity of this optimized solution would be O(n)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/count-ways-form-minimum-product-triplets/

Chapter 43

Count zeros in a row wise and column wise sorted matrix

Count zeros in a row wise and column wise sorted matrix - GeeksforGeeks

Given a N x N binary matrix (elements in matrix can be either 1 or 0) where each row and column of the matrix is sorted in ascending order, count number of 0s present in it.

Expected time complexity is O(N).

Examples:

```
Input:
[0, 0, 0, 0, 1]
[0, 0, 0, 1, 1]
[0, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]

Output: 8
```

```
Input:
[0, 0]
[0, 0]

Output: 4
```

```
Input:
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
```

```
[1, 1, 1, 1]

Output: 0
```

The idea is very simple. We start from the bottom-left corner of the matrix and repeat below steps until we find the top or right edge of the matrix.

1. Decrement row index until we find a 0.
2. Add number of 0s in current column i.e. current row index + 1 to the result and move right to next column (Increment col index by 1).

The above logic will work since the matrix is row-wise and column-wise sorted. The logic will also work for any matrix containing non-negative integers.

Below is the implementation of above idea :

C++

```
 // C++ program to count number of 0s in the given
// row-wise and column-wise sorted binary matrix.
#include <iostream>
using namespace std;
// define size of square matrix
#define N 5

// Function to count number of 0s in the given
// row-wise and column-wise sorted binary matrix.
int countZeroes(int mat[N][N])
{
    // start from bottom-left corner of the matrix
    int row = N - 1, col = 0;

    // stores number of zeroes in the matrix
    int count = 0;

    while (col < N)
    {
        // move up until you find a 0
        while (mat[row][col])

            // if zero is not found in current column,
            // we are done
            if (--row < 0)
                return count;

        // add 0s present in current column to result
        count += (row + 1);

        // move right to next column
        col++;
```

```
    }

    return count;
}

// Driver Program to test above functions
int main()
{
    int mat[N][N] =
    {
        { 0, 0, 0, 0, 1 },
        { 0, 0, 0, 1, 1 },
        { 0, 1, 1, 1, 1 },
        { 1, 1, 1, 1, 1 },
        { 1, 1, 1, 1, 1 }
    };

    cout << countZeroes(mat);

    return 0;
}
```

Java

```
 // Java program to count number of 0s in the given
// row-wise and column-wise sorted binary matrix
import java.io.*;

class GFG
{
    public static int N = 5;

    // Function to count number of 0s in the given
    // row-wise and column-wise sorted binary matrix.
    static int countZeroes(int mat[][])
    {
        // start from bottom-left corner of the matrix
        int row = N - 1, col = 0;

        // stores number of zeroes in the matrix
        int count = 0;

        while (col < N)
        {
            // move up until you find a 0
            while (mat[row][col] > 0)

                // if zero is not found in current column,
```

```
                // we are done
                if (--row < 0)
                    return count;

            // add 0s present in current column to result
            count += (row + 1);

            // move right to next column
            col++;
        }

        return count;
    }

    // Driver program
    public static void main (String[] args)
    {
        int mat[][] = { { 0, 0, 0, 0, 1 },
                        { 0, 0, 0, 1, 1 },
                        { 0, 1, 1, 1, 1 },
                        { 1, 1, 1, 1, 1 },
                        { 1, 1, 1, 1, 1 } };
        System.out.println(countZeroes(mat));
    }
}

// This code is contributed by Pramod Kumar
```

Python

```
 # Python program to count number
# of 0s in the given row-wise
# and column-wise sorted
# binary matrix.

# Function to count number
# of 0s in the given
# row-wise and column-wise
# sorted binary matrix.
def countZeroes(mat):

    # start from bottom-left
    # corner of the matrix
    N = 5;
    row = N - 1;
    col = 0;

    # stores number of
```

```
    # zeroes in the matrix
    count = 0;

    while (col < N):

        # move up until
        # you find a 0
        while (mat[row][col]):

            # if zero is not found
            # in current column, we
            # are done
            if (row < 0):
                return count;
            row = row - 1;

        # add 0s present in
        # current column to result
        count = count + (row + 1);

        # move right to
        # next column
        col = col + 1;

    return count;

# Driver Code
mat = [[0, 0, 0, 0, 1],
       [0, 0, 0, 1, 1],
       [0, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]];

print( countZeroes(mat));

# This code is contributed
# by chandan_jnu
```

C#

```
 // C# program to count number of
// 0s in the given row-wise and
// column-wise sorted binary matrix
using System;

class GFG
{
    public static int N = 5;
```

```
    // Function to count number of
    // 0s in the given row-wise and
    // column-wise sorted binary matrix.
    static int countZeroes(int [,] mat)
    {
        // start from bottom-left
        // corner of the matrix
        int row = N - 1, col = 0;

        // stores number of zeroes
        // in the matrix
        int count = 0;

        while (col < N)
        {
            // move up until you find a 0
            while (mat[row,col] > 0)

                // if zero is not found in
                // current column,
                // we are done
                if (--row < 0)
                    return count;

            // add 0s present in current
            // column to result
            count += (row + 1);

            // move right to next column
            col++;
        }

        return count;
    }

    // Driver Code
    public static void Main ()
    {
        int [,] mat = { { 0, 0, 0, 0, 1 },
                        { 0, 0, 0, 1, 1 },
                        { 0, 1, 1, 1, 1 },
                        { 1, 1, 1, 1, 1 },
                        { 1, 1, 1, 1, 1 } };
        Console.WriteLine(countZeroes(mat));
    }
}
```

```
// This code is contributed by KRV.
```

PHP

```
 <?php
// PHP program to count number
// of 0s in the given row-wise
// and column-wise sorted
// binary matrix.

// Function to count number
// of 0s in the given
// row-wise and column-wise
// sorted binary matrix.
function countZeroes($mat)
{
    // start from bottom-left
    // corner of the matrix
    $N = 5;
    $row = $N - 1;
    $col = 0;

    // stores number of
    // zeroes in the matrix
    $count = 0;

    while ($col < $N)
    {
        // move up until
        // you find a 0
        while ($mat[$row][$col])

            // if zero is not found
            // in current column, we
            // are done
            if (--$row < 0)
                return $count;

        // add 0s present in
        // current column to result
        $count += ($row + 1);

        // move right to
        // next column
        $col++;
    }

    return $count;
```

```
}

// Driver Code
$mat = array(array(0, 0, 0, 0, 1),
             array(0, 0, 0, 1, 1),
             array(0, 1, 1, 1, 1),
             array(1, 1, 1, 1, 1),
             array(1, 1, 1, 1, 1));

echo countZeroes($mat);

// This code is contributed by Sam007
?>
```

Output:

```
8
```

Time complexity of above solution is O(n) since the solution follows single path from bottom-left corner to top or right edge of the matrix.
Auxiliary space used by the program is O(1).

Do share with us if you find more interesting methods of solving this problem.

Improved By : KRV, Sam007, Chandan_Kumar

Source

https://www.geeksforgeeks.org/count-zeros-in-a-row-wise-and-column-wise-sorted-matrix/

Chapter 44

Counting cross lines in an array

Counting cross lines in an array - GeeksforGeeks

Given an unsorted array of distinct elements. Task is to count number of cross lines formed in an array elements after sorting the array elements.
Note: Draw a line between same array elements before sorting and after sorting the array elements.

Examples :

```
Input :  arr[] = { 3, 2, 1, 4, 5 }
Output : 3
      before sort: 3  2  1  4  5
                    \ | /   |  |
                     \|/    |  |
                    / | \   |  |
      After sort : 1  2  3  4  5
      line (1 to 1) cross line (2 to 2)
      line (1 to 1) cross line (3 to 3)
      line (2 to 2) cross line (3 to 3)
Note: the line between two 4s and the line
between two 5s don't cross any other lines;

Input : arr[] = { 5, 4, 3, 1 }
Output : 6
```

Simple solution of this problem is based on the insertion sort. we simply pick each array elements one-by-one and try to find it's proper position in the sorted array.during finding it's appropriate position of an element we have to cross all the element_line whose value is greater than current element.

Below is the implementation of above idea :

C++

```
 // c++ program to count cross line in array
#include <bits/stdc++.h>
using namespace std;

// function return count of cross line in an array
int countCrossLine(int arr[], int n)
{
    int count_crossline = 0;
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        /* Move elements of arr[0..i-1], that are
          greater than key, to one position ahead
          of their current position */
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;

            // increment cross line by one
            count_crossline++;
        }
        arr[j + 1] = key;
    }
    return count_crossline;
}

// driver program to test above function
int main()
{
    int arr[] = { 4, 3, 1, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << countCrossLine(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java program to count
// cross line in array

class GFG
{
    static int countCrossLine(int arr[],
                              int n)
    {
        int count_crossline = 0;
```

```
        int i, key, j;
        for (i = 1; i < n; i++)
        {
            key = arr[i];
            j = i - 1;

            // Move elements of arr[0..i-1],
            // that are greater than key,
            // to one position ahead of
            // their current position
            while (j >= 0 && arr[j] > key)
            {
                arr[j + 1] = arr[j];
                j = j - 1;

                // increment cross
                // line by one
                count_crossline++;
            }
            arr[j + 1] = key;
        }

        return count_crossline;
    }

    // Driver Code
    public static void main(String args[])
    {
        int arr[] = new int[]{ 4, 3, 1, 2 };
        int n = arr.length;
        System.out.print(countCrossLine(arr, n));
    }
}

// This code is contributed by Sam007
```

C#

```
 // C# program to count cross line in array
using System;

class GFG {

    // function return count of cross line
    // in an array
    static int countCrossLine(int []arr, int n)
    {
        int count_crossline = 0;
```

```
        int i, key, j;
        for (i = 1; i < n; i++) {
            key = arr[i];
            j = i - 1;

            /* Move elements of arr[0..i-1],
            that are greater than key, to one
            position ahead of their current
            position */
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;

                // increment cross line by one
                count_crossline++;
            }
            arr[j + 1] = key;
        }

        return count_crossline;
    }

    // Driver code
    public static void Main()
    {
        int []arr = new int[]{ 4, 3, 1, 2 };
        int n = arr.Length;
        Console.Write(countCrossLine(arr, n));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to count
// cross line in array

// Function return count
// of cross line in an array
function countCrossLine($arr, $n)
{

    $count_crossline = 0;
    $i; $key; $j;
    for ($i = 1; $i < $n; $i++)
    {
```

```
        $key = $arr[$i];
        $j = $i - 1;

        /* Move elements of arr[0..i-1], that are
           greater than key, to one position ahead
           of their current position */
        while ($j >= 0 and $arr[$j] > $key)
        {
            $arr[$j + 1] = $arr[$j];
            $j = $j - 1;

            // increment cross line by one
            $count_crossline++;
        }
        $arr[$j + 1] = $key;
    }
    return $count_crossline;
}

    // Driver Code
    $arr = array( 4, 3, 1, 2 );
    $n = count($arr);
    echo countCrossLine($arr, $n);

// This code is contributed by anuj_67.
?>


5
```

Time complexity :O(n^2)
Auxiliary space: O(1)

Efficient solution based on the merge sortand inversions count.

```
   lets we have arr[] { 2, 4, 1, 3 }
                         \   \ /   /
                          \  / \ /
                          / \  / \
   After sort   arr[] { 1, 2, 3, 4 }
   and here all inversion are (2, 1), (4, 1), (4, 3)
   that mean line 1 : cross line 4, 2
             line 2 : cross line 1
             line 4 : cross line 3, 1
             line 3 : cross line 3
 so total unique cross_line are: 3
```

During mer
Below c++ implementation of above idea.

```
 // c++ program to count cross line in array
#include <bits/stdc++.h>
using namespace std;

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r, int* count_crossline)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];

            //====================================//
            //======= MAIN PORTION OF CODE ======//
            //====================================//
            // add all line which is cross by current element
            *count_crossline += (n1 - i);
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
```

```
    are any */
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
    are any */
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* l is for left index and r is right index of the
sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r, int* count_crossline)
{
    if (l < r) {

        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m, count_crossline);
        mergeSort(arr, m + 1, r, count_crossline);

        merge(arr, l, m, r, count_crossline);
    }
}

// function return count of cross line in an array
int countCrossLine(int arr[], int n)
{
    int count_crossline = 0;
    mergeSort(arr, 0, n - 1, &count_crossline);

    return count_crossline;
}

// driver program to test above function
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    cout << countCrossLine(arr, n) << endl;
    return 0;
}
```

Output:

```
10
```

Time complexity :O(nlogn)

Improved By : vt_m, Sam007

Source

https://www.geeksforgeeks.org/counting-cross-lines-array/

Chapter 45

De-arrangements for minimum product sum of two arrays

De-arrangements for minimum product sum of two arrays - GeeksforGeeks

Given two arrays A[] and B[] of same size n. We need to first permute any of arrays such that the sum of product of pairs(1 element from each) is minimum. That is **SUM (Ai*Bi) for all i** is minimum. We also need to count number of de-arrangements present in original array as compared to permuted array.

Examples:

```
Input : A[] = {4, 3, 2},
        B[] = {7, 12, 5}
Output : 3
Explanation : A[] = {4, 3, 2} and B[] = {5, 7, 12}
results in minimum product sum. B[] = {7, 12, 5}
is 3-position different from new B[].

Input : A[] = {4, 3, 2},
        B[] = { 1, 2, 3}
Output : 0
Explanation : A[] = {4, 3, 2} and B[] = {1, 2, 3}
results in minimum product sum. B[] = {1, 2, 3}
is exactly same as new one.
```

Idea behind finding the minimum sum of product from two array is to sort both array one in increasing and other in decreasing manner. These type of arrays will always produce minimum sum of pair product. Sorting both array will give the pair value i.e. which element from A is paired to which element from B[]. After that count the de-arrangement from original arrays.
Algorithm :

1. make a copy of both array.
2. sort copy_A[] in increasing, copy_B[] in decreasing order.
3. Iterate for all Ai, find Ai in copy_A[] as copy_A[j] and
   check whether copy_B[j] == B[i] or not. Increment count if not equal.
4. Return Count Value. That will be our answer.

```
 // CPP program to count de-arrangements for
// minimum product.
#include<bits/stdc++.h>
using namespace std;

// function for finding de-arrangement
int findDearrange (int A[], int B[], int n)
{
    // create copy of array
    vector <int> copy_A (A, A+n);
    vector <int> copy_B (B, B+n);

    // sort array in inc & dec way
    sort(copy_A.begin(), copy_A.end());
    sort(copy_B.begin(), copy_B.end(),greater<int>());

    // count no. of de arrangements
    int count = 0;
    for (int i=0; i<n;i++)
    {
         vector<int>::iterator itA;

         // find position of A[i] in sorted array
         itA = lower_bound(copy_A.begin(),
                           copy_A.end(), A[i]);

         // check whether B[i] is same as required or not
         if (B[i] != copy_B[itA-copy_A.begin()])
              count++;
    }

    // return count
    return count;
}

// driver function
int main()
{
    int A[] = {1, 2, 3, 4};
    int B[] = {6, 3, 4, 5};
    int n = sizeof(A) /sizeof(A[0]);;
    cout << findDearrange(A,B,n);
```

```
    return 0;
}
```

Output:

```
2
```

Source

https://www.geeksforgeeks.org/de-arrangements-for-minimum-product-sum-of-two-arrays/

Chapter 46

Delete array element in given index range [L – R]

Delete array element in given index range [L - R] - GeeksforGeeks

Given an array A[] and size of array is N. The task is to delete element of array A[] are in given range L to R both are exclusive.

Examples:

```
Input : N = 12
        A[] = { 3, 5, 3, 4, 9, 3, 1, 6, 3, 11, 12, 3}
        L = 2
        R = 7
Output : 3 5 3 6 3 11 12 3
since A[2] = 3 but this is exclude
A[7] =  6 this also exclude

Input : N = 10
        A[] ={ 5, 8, 11, 15, 26, 14, 19, 17, 10, 14 }
        L = 4
        R = 6
Output :5 8 11 15 26 19 17 10 14
```

A **naive approach** is to delete element in range L to R with extra space.

Below is the implementation of the above approach:

```
 // C++ code to delete element
// in given range
#include <bits/stdc++.h>
using namespace std;
```

```
// Delete L to R element
vector<int> deleteElement(int A[], int L, int R, int N)
{
    vector<int> B;

    for (int i = 0; i < N; i++)
        if (i <= L || i >= R)
           B.push_back(A[i]);

    return B;
}

// main Driver
int main()
{
    int A[] = { 3, 5, 3, 4, 9, 3, 1, 6, 3, 11, 12, 3 };
    int L = 2, R = 7;
    int n = sizeof(A) / sizeof(A[0]);
    vector<int> res = deleteElement(A, L, R, n);
    for (auto x : res)
        cout << x << " ";
    return 0;
}
```

Output:

```
3 5 3 6 3 11 12 3
```

Time Complexity: O(n)
Auxiliary Space : O(n)

An **efficient solution** without using extra space.
Below is the implementation of the above approach:

C++

```
 // C++ code to delete element
// in given range
#include <bits/stdc++.h>
using namespace std;

// Delete L to R elements
int deleteElement(int A[], int L, int R, int N)
{
    int i, j = 0;
    for (i = 0; i < N; i++) {
```

```
        if (i <= L || i >= R) {
            A[j] = A[i];
            j++;
        }
    }

    // Return size of Array
    // after delete element
    return j;
}

// main Driver
int main()
{
    int A[] = { 5, 8, 11, 15, 26, 14, 19, 17, 10, 14 };
    int L = 2, R = 7;
    int n = sizeof(A) / sizeof(A[0]);
    int res_size = deleteElement(A, L, R, n);
    for (int i = 0; i < res_size; i++)
        cout << A[i] << " ";
    return 0;
}
```

Java

```
 // Java code to delete element
// in given range
class GFG
{

// Delete L to R elements
static int deleteElement(int A[], int L,
                         int R, int N)
{
    int i, j = 0;
    for (i = 0; i < N; i++)
    {
        if (i <= L || i >= R)
        {
            A[j] = A[i];
            j++;
        }
    }

    // Return size of Array
    // after delete element
    return j;
}
```

```
// Driver Code
public static void main(String args[])
{
    int A[] = new int[] { 5, 8, 11, 15, 26,
                          14, 19, 17, 10, 14 };
    int L = 2, R = 7;
    int n = A.length;
    int res_size = deleteElement(A, L, R, n);
    for (int i = 0; i < res_size; i++)
    System.out.print(A[i] + " ");
}
}

// This code is contributed
// by Kirti_Mangal
```

Python 3

```
 # Python 3 program  to delete element
# in given range

# Function to delete L to R element
def deleteElement(A, L, R, N) :

    j = 0
    for i in range(N) :
        if i <= L or i >= R :
            A[j] = A[i]
            j += 1

    # Return size of Array
    # after delete element
    return j

# Driver Code
if __name__ == "__main__" :

    A = [5, 8, 11, 15, 26, 14, 19, 17, 10, 14]
    L, R = 2,7

    n = len(A)
    res_size = deleteElement(A, L, R, n)

    for i in range(res_size) :
        print(A[i],end = " ")

# This code is contributed by ANKITRAI1
```

Output:

```
5 8 11 17 10 14
```

Time Complexity: O(n)
Auxiliary Space : O(1)

Improved By : Kirti_Mangal, ANKITRAI1

Source

https://www.geeksforgeeks.org/delete-array-element-in-given-index-range-l-r/

Chapter 47

Divide an array into k segments to maximize maximum of segment minimums

Divide an array into k segments to maximize maximum of segment minimums - GeeksforGeeks

Given an array of n integers, divide it into k segments and find the maximum of the minimums of k segments. Output the maximum integer that can be obtained among all ways to segment in k subarrays.

Examples:

```
Input : arr[] = {1, 2, 3, 6, 5}
        k = 2
Output: 5
Explanation: There are many ways to create
two segments. The optimal segments are (1, 2, 3)
and (6, 5). Minimum of both segments are 1 and 5,
hence the maximum(1, 5) is 5.
```

```
Input: -4 -5 -3 -2 -1 k=1
Output: -5
Explanation: only one segment, so minimum is -5.
```

There will be 3 cases that need to be considered.

1. **k >= 3:** When k is greater then 2, one segment will only compose of {max element}, so that max of minimum segments will always be the max.

2. **k = 2:** For k = 2 the answer is the maximum of the first and last element.
3. **k = 1:** Only possible partition is one segment equal to the whole array. So the answer is the minimum value on the whole array.

Below is the implementation of the above approach

C++

```
 // CPP Program to find maximum value of
// maximum of minimums of k segments.
#include <bits/stdc++.h>
using namespace std;

// function to calculate the max of all the
// minimum segments
int maxOfSegmentMins(int a[], int n, int k)
{
    // if we have to divide it into 1 segment
    // then the min will be the answer
    if (k == 1)
       return *min_element(a, a+n);

    if (k == 2)
       return max(a[0], a[n-1]);

    // If k >= 3, return maximum of all
    // elements.
    return *max_element(a, a+n);
}

// driver program to test the above function
int main()
{
    int a[] = { -10, -9, -8, 2, 7, -6, -5 };
    int n = sizeof(a) / sizeof(a[0]);
    int k = 2;
    cout << maxOfSegmentMins(a, n, k);
}
```

Java

```
 // Java Program to find maximum
// value of maximum of minimums
// of k segments.
import java .io.*;
import java .util.*;

class GFG
```

```
{

    // function to calculate
    // the max of all the
    // minimum segments
    static int maxOfSegmentMins(int []a,
                                int n,
                                int k)
    {

        // if we have to divide
        // it into 1 segment then
        // the min will be the answer
        if (k == 1)
        {
            Arrays.sort(a);
                return a[0];
        }

        if (k == 2)
            return Math.max(a[0],
                            a[n - 1]);

        // If k >= 3, return
        // maximum of all
        // elements.
        return a[n - 1];
    }

    // Driver Code
    static public void main (String[] args)
    {
        int []a = {-10, -9, -8,
                    2, 7, -6, -5};
        int n = a.length;
        int k = 2;

        System.out.println(
                maxOfSegmentMins(a, n, k));
    }
}

// This code is contributed
// by anuj_67.
```

C#

```
 // C# Program to find maximum value of
```

```
// maximum of minimums of k segments.
using System;
using System.Linq;

public class GFG {


    // function to calculate the max
    // of all the minimum segments
    static int maxOfSegmentMins(int []a,
                            int n, int k)
    {

        // if we have to divide it into 1
        // segment then the min will be
        // the answer
        if (k == 1)
            return a.Min();

        if (k == 2)
            return Math.Max(a[0], a[n - 1]);

        // If k >= 3, return maximum of
        // all elements.
        return a.Max();
    }

    // Driver function
    static public void Main ()
    {
        int []a = { -10, -9, -8, 2, 7,
                                -6, -5 };
        int n = a.Length;
        int k = 2;

        Console.WriteLine(
                  maxOfSegmentMins(a, n, k));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find maximum
// value of maximum of minimums
// of k segments.
```

```
// function to calculate
// the max of all the
// minimum segments
function maxOfSegmentMins($a, $n, $k)
{
    // if we have to divide
    // it into 1 segment then
    // the min will be the answer
    if ($k == 1)
    return min($a);

    if ($k == 2)
    return max($a[0],
               $a[$n - 1]);

    // If k >= 3, return
    // maximum of all elements.
    return max($a);
}

// Driver Code
$a = array(-10, -9, -8,
            2, 7, -6, -5);
$n = count($a);
$k = 2;
echo maxOfSegmentMins($a, $n, $k);

// This code is contributed by vits.
?>
```

Output:

```
-5
```

Time complexity: O(n)
Auxiliary Space: O(1)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/divide-array-k-segments-maximize-maximum-segment-minimums/

Chapter 48

Duplicates in an array in O(n) time and by using O(1) extra space | Set-3

Duplicates in an array in O(n) time and by using O(1) extra space | Set-3 - GeeksforGeeks

Given an array of n elements which contains elements from 0 to n-1, with any of these numbers appearing any number of times. Find these repeating numbers in O(n) and using only constant memory space. It is required that the order in which elements repeat should be maintained. If there is no repeating element present then print -1.

Examples:

```
Input : arr[] = {1, 2, 3, 1, 3, 6, 6}
Output : 1, 3, 6
Elements 1, 3 and 6 are repeating.
Second occurrence of 1 is found
first followed by repeated occurrence
of 3 and 6.

Input : arr[] = {0, 3, 1, 3, 0}
Output : 3, 0
Second occurrence of 3 is found
first followed by second occurrence
of 0.
```

We have discussed two approaches for this question in below posts:
Find duplicates in O(n) time and O(1) extra space | Set 1
Duplicates in an array in O(n) time and by using O(1) extra space | Set-2

There is a problem in first approach. It prints the repeated number more than once. For example: {1, 6, 3, 1, 3, 6, 6} it will give output as : 3 6 6. In second approach although each repeated item is printed only once, the order in which their repetition occurs is not maintained. To print elements in order in which they are repeating, the second approach is modified. To mark the presence of an element size of the array, n is added to the index position arr[i] corresponding to array element arr[i]. Before adding n, check if value at index arr[i] is greater than or equal to n or not. If it is greater than or equal to, then this means that element arr[i] is repeating. To avoid printing repeating element multiple times, check if it is the first repetition of arr[i] or not. It is first repetition if value at index position arr[i] is less than 2*n. This is because, if element arr[i] has occurred only once before then n is added to index arr[i] only once and thus value at index arr[i] is less than 2*n. Add n to index arr[i] so that value becomes greater than or equal to 2*n and it will prevent further printing of current repeating element. Also if value at index arr[i] is less than n then it is first occurrence (not repetition) of element arr[i]. So to mark this add n to element at index arr[i].

Below is the implementation of above approach:

```
 // C++ program to print all elements that
// appear more than once.
#include <bits/stdc++.h>
using namespace std;

// Function to find repeating elements
void printDuplicates(int arr[], int n)
{
    int i;

    // Flag variable used to
    // represent whether repeating
    // element is found or not.
    int fl = 0;

    for (i = 0; i < n; i++) {

        // Check if current element is
        // repeating or not. If it is
        // repeating then value will
        // be greater than or equal to n.
        if (arr[arr[i] % n] >= n) {

            // Check if it is first
            // repetition or not. If it is
            // first repetition then value
            // at index arr[i] is less than
            // 2*n. Print arr[i] if it is
            // first repetition.
            if (arr[arr[i] % n] < 2 * n) {
                cout << arr[i] % n << " ";
```

```
                fl = 1;
            }
        }

        // Add n to index arr[i] to mark
        // presence of arr[i] or to
        // mark repetition of arr[i].
        arr[arr[i] % n] += n;
    }

    // If flag variable is not set
    // then no repeating element is
    // found. So print -1.
    if (!fl)
        cout << "-1";
}

// Driver Function
int main()
{
    int arr[] = { 1, 6, 3, 1, 3, 6, 6 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);
    printDuplicates(arr, arr_size);
    return 0;
}
```

Output:

```
1 3 6
```

Time Complexity: O(n)
Auxiliary Space: O(1)

Source

https://www.geeksforgeeks.org/duplicates-in-an-array-in-on-time-and-by-using-o1-extra-space-set-3/

Chapter 49

Efficient search in an array where difference between adjacent is 1

Efficient search in an array where difference between adjacent is 1 - GeeksforGeeks

Given an array of **n** integers. Each array element is obtained by adding either +1 or -1 to previous element i.e absolute difference between any two consecutive elements is 1. The task is to search an element index with the minimum number of comparison (less than simple element by element search). If the element is present multiple time, then print the smallest index. If the element is not present print -1.

Examples:

```
Input : arr[] = {5, 4, 5, 6, 5, 4, 3, 2}
        x = 4.
Output : 1
The first occurrence of element x is at
index 1.

Input : arr[] = { 5, 4, 5, 6, 4, 3, 2, 3 }
        x = 9.
Output : -1
Element x is not present in arr[]
```

Let element to be search is x. At any index i, if arr[i] != x, the possibility of x to be present is at location i + abs(arr[i] – a), since each element is obtained by adding either +1 or -1 to the previous element. There is no possibility of having el between i and i + abs(arr[i] – a). So directly jump to i + abs(arr[i] – a), if arr[i] != x.

```
Algorithm to solve the problem:
1. Start from index = 0.
2. Compare arr[index] and x.
   a) If both are equal, return index.
   b) If not, set index = index + abs(arr[index] - x).
3. Repeat step 2.
```

Below is the implementation of above idea :

C++

```
 // C++ program to search an element in an array
// where each element is obtained by adding
// either +1 or -1 to previous element.
#include<bits/stdc++.h>
using namespace std;

// Return the index of the element to be searched.
int search(int arr[], int n, int x)
{
    // Searching x in arr[0..n-1]
    int i = 0;
    while (i <= n-1)
    {
        // Checking if element is found.
        if (arr[i] == x)
            return i;

        // Else jumping to abs(arr[i]-x).
        i += abs(arr[i]-x);
    }

    return -1;
}

// Driven Program
int main()
{
    int arr[] =  {5, 4, 5, 6, 5, 4, 3, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 4;

    cout << search(arr, n, x) << endl;

    return 0;
}
```

Java

```
 // Java program to search an element
// in an array where each element is
// obtained by adding either +1 or
// -1 to previous element.
class GFG
{

// Return the index of the
// element to be searched.
static int search(int arr[], int n, int x)
{
    // Searching x in arr[0..n-1]
    int i = 0;
    while (i <= n-1)
    {
        // Checking if element is found.
        if (arr[i] == x)
            return i;

        // Else jumping to abs(arr[i]-x).
        i += Math.abs(arr[i]-x);
    }

    return -1;
}

// Driver code
public static void main (String[] args)
{
    int arr[] = {5, 4, 5, 6, 5, 4, 3, 2};
    int n = arr.length;
    int x = 4;

    System.out.println(search(arr, n, x));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python program to search an element in
# an array where each element is obtained
# by adding either +1 or -1 to previous element
```

```
# Return the index of the element to be searched
def search(arr, n, x):

    # Searching x in arr[0..n-1]
    i = 0
    while (i <= n-1):

        # Checking if element is found.
        if (arr[i] == x):
            return i

        # Else jumping to abs(arr[i]-x).
        i += abs(arr[i] - x)

    return -1

# Driver code
arr = [5, 4, 5, 6, 5, 4, 3, 2]
n = len(arr)
x = 4

print(search(arr, n, x))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to search an element in
// an array where each element is
// obtained by adding either + 1 or
// -1 to previous element.
using System;

class GFG
{

// Return the index of the
// element to be searched.
static int search(int []arr, int n,
                  int x)
{

    // Searching x in arr[0.. n - 1]
    int i = 0;
    while (i <= n - 1)
    {
        // Checking if element is found
        if (arr[i] == x)
```

```
            return i;

        // Else jumping to abs(arr[i] - x)
        i += Math.Abs(arr[i] - x);
    }

    return -1;
}

// Driver code
public static void Main ()
{
    int []arr = {5, 4, 5, 6, 5, 4, 3, 2};
    int n = arr.Length;
    int x = 4;

    Console.WriteLine(search(arr, n, x));
}
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to search an
// element in an array where
// each element is obtained
// by adding either +1 or -1
// to previous element.

// Return the index of the
// element to be searched.
function search($arr, $n, $x)
{

    // Searching x in arr[0..n-1]
    $i = 0;
    while ($i <= $n-1)
    {

        // Checking if element
        // is found.
        if ($arr[$i] == $x)
            return $i;

        // Else jumping to
        // abs(arr[i]-x).
```

```
        $i += abs($arr[$i] - $x);
    }

    return -1;
}

    // Driver Code
    $arr= array(5, 4, 5, 6, 5, 4, 3, 2);
    $n = sizeof($arr);
    $x = 4;

    echo search($arr, $n, $x) ;

// This code is contributed by nitin mittal.
?>
```

Output:

```
1
```

Improved By : vt_m, nitin mittal

Source

https://www.geeksforgeeks.org/efficient-search-in-an-array-where-difference-between-adjacent-is-1/

Chapter 50

Even-odd turn game with two integers

Even-odd turn game with two integers - GeeksforGeeks

Given three positive integers X, Y and P. Here P denotes the number of turns. Whenever the turn is odd X is multiplied by 2 and in every even turn Y is multiplied by 2. The task is to find the value of max(X, Y) ÷ min(X, Y) after the complete P turns.

Examples :

```
Input : X = 1, Y = 2, P = 1
Output : 1
As turn is odd, X is multiplied by
2 and becomes 2. Now, X is 2 and Y is also 2.
Therefore, 2 ÷ 2 is 1.

Input : X = 3, Y = 7, p = 2
Output : 2
Here we have 2 turns. In the 1st turn which is odd
X is multiplied by 2. And the values are 6 and 7.
In the next turn which is even Y is multiplied by 2.
Now the final values are 6 and 14. Therefore, 14 ÷ 6 is 2.
```

Lets play the above game for 8 turns :

```
| i    | 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7   | 8   |
|------|---|----|----|----|----|----|----|-----|-----|
| X(i) | X | 2X | 2X | 4X | 4X | 8X | 8X | 16X | 16X |
| Y(i) | Y | Y  | 2Y | 2Y | 4Y | 4Y | 8Y | 8Y  | 16Y |
```

Here we can easily spot a pattern :

```
if i is even, then X(i) = z * X and Y(i) = z * Y.
if i is odd, then X(i) = 2*z * X and Y(i) = z * Y.
```

Here z is actually the power of 2. So, we can simply say –

```
If P is even output will be max(X, Y) ÷ min(X, Y)
else output will be max(2*X, Y) ÷ min(2*X, Y).
```

Below is the implementation :

C++

```
 // CPP program to find max(X, Y) / min(X, Y)
// after P turns
#include <bits/stdc++.h>
using namespace std;

int findValue(int X, int Y, int P)
{
    if (P % 2 == 0)
        return (max(X, Y) / min(X, Y));

    else
        return (max(2 * X, Y) / min(2 * X, Y));
}

// Driver code
int main()
{
    // 1st test case
    int X = 1, Y = 2, P = 1;
    cout << findValue(X, Y, P) << endl;

    // 2nd test case
    X = 3, Y = 7, P = 2;
    cout << findValue(X, Y, P) << endl;
}
```

Java

```
 // Java program to find max(X, Y) / min(X, Y)
// after P turns
import java.util.*;
```

```
class Even_odd{
    public static int findValue(int X, int Y,
                                        int P)
    {
        if (P % 2 == 0)
            return (Math.max(X, Y) /
                            Math.min(X, Y));

        else
            return (Math.max(2 * X, Y) /
                            Math.min(2 * X, Y));
    }

    public static void main(String[] args)
    {
        // 1st test case
        int X = 1, Y = 2, P = 1;
        System.out.println(findValue(X, Y, P));

        // 2nd test case
        X = 3;
        Y = 7;
        P = 2;
        System.out.print(findValue(X, Y, P));
    }
}

//This code is contributed by rishabh_jain
```

Python3

```
 # Python3 code to find max(X, Y) / min(X, Y)
# after P turns

def findValue( X , Y , P ):
    if P % 2 == 0:
        return int(max(X, Y) / min(X, Y))

    else:
        return int(max(2 * X, Y) / min(2 * X, Y))

# Driver code
# 1st test case
X = 1
Y = 2
P = 1
print(findValue(X, Y, P))
```

```
# 2nd test case
X = 3
Y = 7
P = 2
print((findValue(X, Y, P)))

# This code is contribted by "Sharad_Bhardwaj".
```

C#

```
 // C# program to find max(X, Y) / min(X, Y)
// after P turns
using System;

class GFG
{
    public static int findValue(int X, int Y,
                                        int P)
    {
        if (P % 2 == 0)
            return (Math.Max(X, Y) /
                    Math.Min(X, Y));

        else
            return (Math.Max(2 * X, Y) /
                    Math.Min(2 * X, Y));
    }

    // Driver code
    public static void Main()
    {
        // 1st test case
        int X = 1, Y = 2, P = 1;
        Console.WriteLine(findValue(X, Y, P));

        // 2nd test case
        X = 3;
        Y = 7;
        P = 2;
        Console.WriteLine(findValue(X, Y, P));
    }
}

//This code is contributed by vt_m
```

PHP

```
 <?php
```

```
// PHP program to find
// max(X, Y) / min(X, Y)
// after P turns

function findValue($X, $Y, $P)
{
    if ($P % 2 == 0)
        return (int)(max($X, $Y) /
                     min($X, $Y));

    else
        return (int)(max(2 * $X, $Y) /
                     min(2 * $X, $Y));
}

// Driver code

// 1st test case
$X = 1;
$Y = 2;
$P = 1;
echo findValue($X, $Y, $P), "\n";

// 2nd test case
$X = 3; $Y = 7; $P = 2;
echo findValue($X, $Y, $P), "\n";

// This code is contributed by ajit
?>
```

Output:

```
1
2
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/even-odd-turn-game-with-two-integers/

Chapter 51

Exponential Search

Exponential Search - GeeksforGeeks

The name of this searching algorithm may be misleading as it works in O(Log n) time. The name comes from the way it searches an element.

```
Given a sorted array, and an element x to be
searched, find position of x in the array.

Input:  arr[] = {10, 20, 40, 45, 55}
        x = 45
Output: Element found at index 3

Input:  arr[] = {10, 15, 25, 45, 55}
        x = 15
Output: Element found at index 1
```

We have discussed, linear search, binary search for this problem.

Exponential search involves two steps:

1. Find range where element is present
2. Do Binary Search in above found range.

How to find the range where element may be present?
The idea is to start with subarray size 1, compare its last element with x, then try size 2, then 4 and so on until last element of a subarray is not greater.
Once we find an index i (after repeated doubling of i), we know that the element must be present between i/2 and i (Why i/2? because we could not find a greater value in previous iteration)

Given below are the implementations of above steps.

C++

```
 // C++ program to find an element x in a
// sorted array using Exponential search.
#include <bits/stdc++.h>
using namespace std;

int binarySearch(int arr[], int, int, int);

// Returns position of first ocurrence of
// x in array
int exponentialSearch(int arr[], int n, int x)
{
    // If x is present at firt location itself
    if (arr[0] == x)
        return 0;

    // Find range for binary search by
    // repeated doubling
    int i = 1;
    while (i < n && arr[i] <= x)
        i = i*2;

    //  Call binary search for the found range.
    return binarySearch(arr, i/2, min(i, n), x);
}

// A recursive binary search function. It returns
// location of x in  given array arr[l..r] is
// present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - l)/2;

        // If the element is present at the middle
        // itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then it
        // can only be present n left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid-1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid+1, r, x);
    }
```

```
    // We reach here when element is not present
    // in array
    return -1;
}

// Driver code
int main(void)
{
   int arr[] = {2, 3, 4, 10, 40};
   int n = sizeof(arr)/ sizeof(arr[0]);
   int x = 10;
   int result = exponentialSearch(arr, n, x);
   (result == -1)? printf("Element is not present in array")
                 : printf("Element is present at index %d",
                                                    result);
   return 0;
}
```

Java

```
 // Java     program to find an element x in a
// sorted array using Exponential search.

import java.util.Arrays;

class GFG
{
    // Returns position of first occurrence of
    // x in array
    static int exponentialSearch(int arr[],
                                 int n, int x)
    {
        // If x is present at firt location itself
        if (arr[0] == x)
            return 0;

        // Find range for binary search by
        // repeated doubling
        int i = 1;
        while (i < n && arr[i] <= x)
            i = i*2;

        // Call binary search for the found range.
        return Arrays.binarySearch(arr, i/2,
                                   Math.min(i, n), x);
    }
```

```
    // Driver code
    public static void main(String args[])
    {
        int arr[] = {2, 3, 4, 10, 40};
        int x = 10;
        int result = exponentialSearch(arr, arr.length, x);

        System.out.println((result < 0) ?
                           "Element is not present in array" :
                           "Element is present at index " +
                            result);
    }
}
```

Python

```
 # Python program to find an element x
# in a sorted array using Exponential Search

# A recurssive binary search function returns
# location  of x in given array arr[l..r] is
# present, otherwise -1
def binarySearch( arr, l, r, x):
    if r >= l:
        mid = l + ( r-l ) / 2

        # If the element is present at
        # the middle itself
        if arr[mid] == x:
            return mid

        # If the element is smaller than mid,
        # then it can only be present in the
        # left subarray
        if arr[mid] > x:
            return binarySearch(arr, l,
                                mid - 1, x)

        # Else he element can only be
        # present in the right
        return binarySearch(arr, mid + 1, r, x)

    # We reach here if the element is not present
    return -1

# Returns the position of first
# occurence of x in array
def exponentialSearch(arr, n, x):
```

```
    # IF x is present at first
    # location itself
    if arr[0] == x:
        return 0

    # Find range for binary search
    # j by repeated doubling
    i = 1
    while i < n and arr[i] <= x:
        i = i * 2

    # Call binary search for the found range
    return binarySearch( arr, i / 2,
                         min(i, n), x)


# Driver Code
arr = [2, 3, 4, 10, 40]
n = len(arr)
x = 10
result = exponentialSearch(arr, n, x)
if result == -1:
    print "Element not found in thye array"
else:
    print "Element is present at index %d" %(result)

# This code is contributed by Harshit Agrawal
```

C#

```
 // C# program to find an element x in a
// sorted array using Exponential search.
using System;
class GFG {

// Returns position of first
// ocurrence of x in array
static int exponentialSearch(int []arr,
                         int n, int x)
{

    // If x is present at
    // first location itself
    if (arr[0] == x)
        return 0;

    // Find range for binary search
    // by repeated doubling
```

```
    int i = 1;
    while (i < n && arr[i] <= x)
        i = i * 2;

    // Call binary search for
    // the found range.
    return binarySearch(arr, i/2,
                        Math.Min(i, n), x);
}

// A recursive binary search
// function. It returns location
// of x in given array arr[l..r] is
// present, otherwise -1
static int binarySearch(int []arr, int l,
                        int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - l) / 2;

        // If the element is present
        // at the middle itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than
        // mid, then it can only be
        // present n left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        // Else the element can only
        // be present in right subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // We reach here when element
    // is not present in array
    return -1;
}

// Driver code
public static void Main()
{
    int []arr = {2, 3, 4, 10, 40};
    int n = arr.Length;
    int x = 10;
```

```
    int result = exponentialSearch(arr, n, x);
    if (result == -1)
            Console.Write("Element is not present in array");
        else
            Console.Write("Element is present at index "
                                                + result);
}
}

// This code is contributed by Smitha
```

PHP

```
 <?php
// PHP program to find an element x in a
// sorted array using Exponential search.

// Returns position of first
// ocurrence of x in array
function exponentialSearch($arr, $n, $x)
{

    // If x is present at
    // first location itself
    if ($arr[0] == $x)
        return 0;

    // Find range for binary search
    // by repeated doubling
    $i = 1;
    while ($i< $n and $arr[$i] <=$x)
        $i = $i * 2;

    // Call binary search
    // for the found range.
    return binarySearch($arr, $i / 2,
                        min($i, $n), $x);
}

// A recursive binary search
// function. It returns location
// of x in given array arr[l..r] is
// present, otherwise -1
function binarySearch($arr, $l,
                      $r, $x)
{
    if ($r >= $l)
    {
```

```
        $mid = $l + ($r - $l)/2;

        // If the element is
        // present at the middle
        // itself
        if ($arr[$mid] == $x)
            return $mid;

        // If element is smaller
        // than mid, then it
        // can only be present
        // n left subarray
        if ($arr[$mid] > $x)
            return binarySearch($arr, $l,
                                $mid - 1, $x);

        // Else the element
        // can only be present
        // in right subarray
        return binarySearch($arr, $mid + 1,
                                    $r, $x);
    }

    // We reach here when
    // element is not present
    // in array
    return -1;
}

// Driver code
$arr = array(2, 3, 4, 10, 40);
$n = count($arr);
$x = 10;
$result = exponentialSearch($arr, $n, $x);
if($result == -1)
    echo "Element is not present in array";
else
    echo "Element is present at index ",
                                $result;

// This code is contributed by anuj_67
?>
```

Output :

```
Element is present at index 3
```

Time Complexity : O(Log n)

Auxiliary Space : The above implementation of Binary Search is recursive and requires O(Log n) space. With iterative Binary Search, we need only O(1) space.

Applications of Exponential Search:

1. Exponential Binary Search is particularly useful for unbounded searches, where size of array is infinite. Please refer Unbounded Binary Search for an example.
2. It works better than Binary Search for bounded arrays, and also when the element to be searched is closer to the first element.

Reference:
https://en.wikipedia.org/wiki/Exponential_search

Improved By : Anamitra Musib, vt_m, Smitha Dinesh Semwal, vsushko

Source

https://www.geeksforgeeks.org/exponential-search/

Chapter 52

Extended Mo's Algorithm with O(1) time complexity

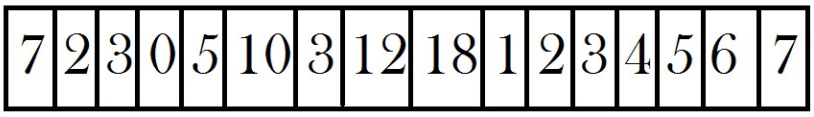
Extended Mo's Algorithm with O(1) time complexity - GeeksforGeeks

Given an array of n elements and q range queries (range sum in this article) with no updates, task is to answer these queries with efficient time and space complexity. The time complexity of a range query after applying square root decomposition comes out to be **O(√n)**. This square-root factor can be decreased to a constant linear factor by applying square root decomposition on the block of the array which was decomposed earlier.

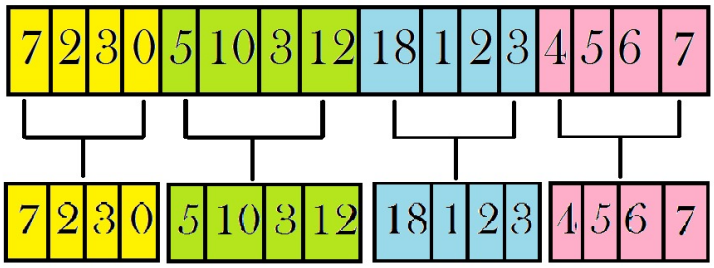
Prerequisite: Mo's Algorithm | Prefix Array

Approach :
As we apply square root decomposition to the given array, querying a range-sum comes in **O(√n)** time.
Here, calculate the sum of blocks which are in between the blocks under consideration(corner blocks), which takes **O(√n)** iterations.

Initial Array :



Decomposition of array into blocks :



DECOMPOSED ARRAY

And the calculation time for the sum on the starting block and ending block both takes **O(√n)** iterations.
Which will leaves us per query time complexity of :

```
= O(√n) + O(√n) + O(√n)
= 3 * O(√n)
~O(√n)
```

Here, we can reduce the runtime complexity of our query algorithm cleverly by calculating blockwise prefix sum and using it to calculate the sum accumulated in the blocks which lie between the blocks under consideration. Consider the code below :

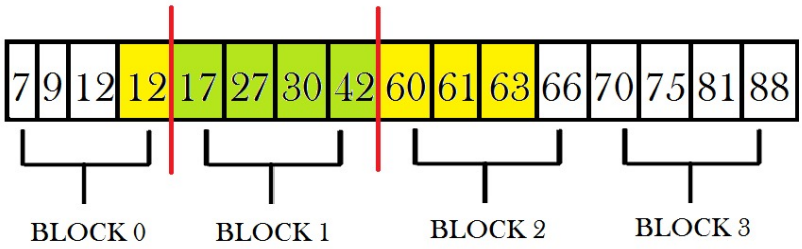
```
interblock_sum[x1][x2] = prefixData[x2 - 1] - prefixData[x1];
```

Time taken for calculation of above table is :

```
= O(√n) * O(√n)
~O(n)
```

NOTE : We haven't taken the sum of blocks x1 & x2 under consideration as they might be carrying partial data.

Prefix Array :



Suppose we want to query for the sum for range from 4 to 11, we consider the sum between block 0 and block 2 (excluding the data contained in block 0 and block 1), which can be calculated using the sum in the green coloured blocks represented in the above image.

Sum between block 0 and block 2 = 42 – 12 = 30

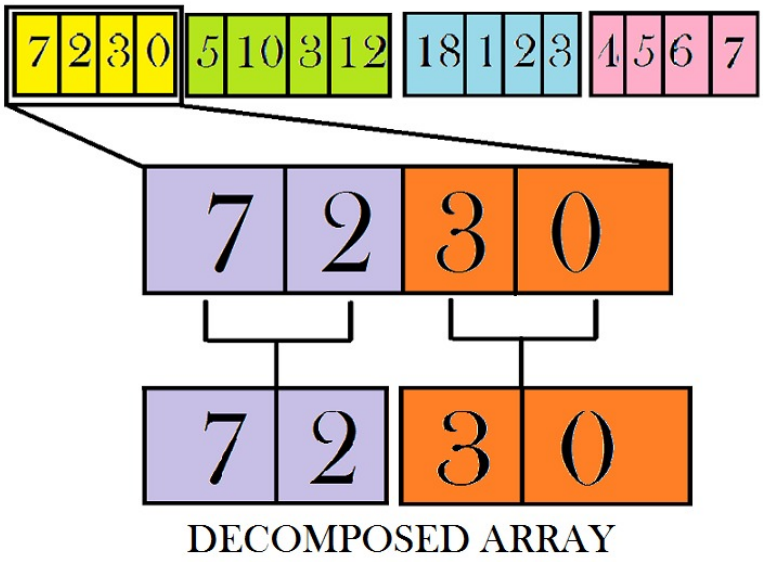
For calculation of rest of the sum present in the yellow blocks, consider the prefix array at the decomposition level-2 and repeat the process again.

Here, observe that we have reduced our time complexity per query significantly, though our runtime remains similar to our last approach :

Our new time complexity can be calculated as :

```
= O(√n) + O(1) + O(√n)
= 2 * O(√n)
~O(√n)
```

Square-root Decomposition at Level-2 :



Further, we apply square root decomposition again on every decomposed block retained from the previous decomposition. Now at this level, we have approximately **√√n** sub-blocks in each block which were decomposed at last level. So, we need to run a range query on these blocks only two times, one time for starting block and one time for ending block.

Precalcuation Time taken for level 2 decomposition :
No of blocks at level 1 ~ **√n**
No of blocks at level 2 ~ **√√n**

Level-2 Decomposition Runtime of a level-1 decomposed block :

```
= O(√n)
```

Overall runtime of level-2 decomposition over all blocks :

```
= O(√n) * O(√n)
~O(n)
```

Now, we can query our level-2 decomposed blocks in **O(√√n)** time.
So, we have reduced our overall time complexity from **O(√n)** to **O(√√n)**

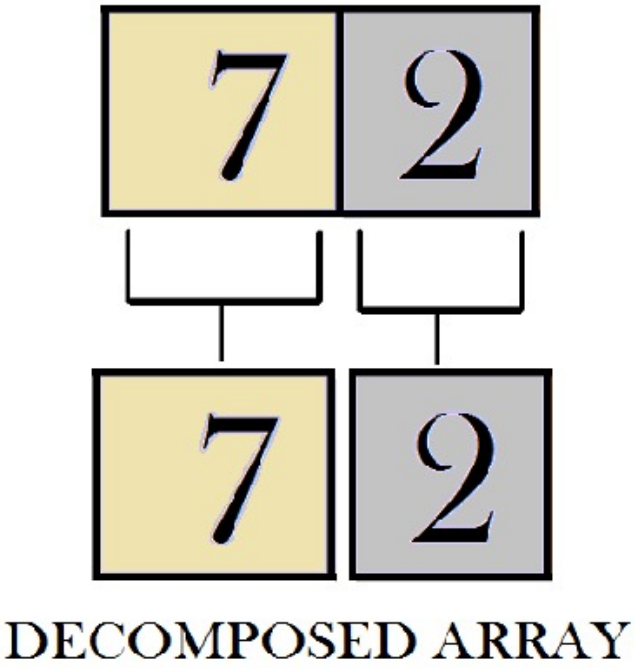
Time complxity taken in querying edge blocks :

```
= O(√√n) + O(1) + O(√√n)
= 2 * O(√√n)
~O(√√n)
```

Total Time complexity can be calculated as :

```
= O(√√n)+O(1)+O(√√n)
= 2 * O(√√n)
~O(√√n)
```

Square-root Decomposition at Level-3 :



DECOMPOSED ARRAY

Using this method we can decompose our array again and again recursively **d times** to reduce our time complexity to a factor of **constant linearity**.

```
O(d * n1/(2^d)) ~ O(k), as d increases this factor converges to a constant linear term
```

The code presented below is a representation of triple square root decomposition where d = 3:

```
O(q * d * n1/(2 ^ 3)) ~ O(q * k) ~ O(q)
```

[where q repesents number of range queries]

```
 // CPP code for offline queries in
// approx constant time.
#include<bits/stdc++.h>
using namespace std;

int n1;

// Structure to store decomposed data
typedef struct
{
    vector<int> data;
    vector<vector<int>> rdata;
    int blocks;
    int blk_sz;
}sqrtD;

vector<vector<sqrtD>> Sq3;
vector<sqrtD> Sq2;
sqrtD Sq1;

// Square root Decomposition of
// a given array
sqrtD decompose(vector<int> arr)
{
    sqrtD sq;
    int n = arr.size();
    int blk_idx = -1;
    sq.blk_sz = sqrt(n);
    sq.data.resize((n/sq.blk_sz) + 1, 0);

    // Calculation of data in blocks
    for (int i = 0; i < n; i++)
    {
        if (i % sq.blk_sz == 0)
        {
            blk_idx++;
        }
        sq.data[blk_idx] += arr[i];
    }

    int blocks = blk_idx + 1;
    sq.blocks = blocks;

    // Calculation of prefix data
    int prefixData[blocks];
    prefixData[0] = sq.data[0];
    for(int i = 1; i < blocks; i++)
    {
```

```
        prefixData[i] =
          prefixData[i - 1] + sq.data[i];
    }

    sq.rdata.resize(blocks + 1,
             vector<int>(blocks + 1));

    // Calculation of data between blocks
    for(int i = 0 ;i < blocks; i++)
    {
        for(int j = i + 1; j < blocks; j++)
        {
            sq.rdata[i][j] = sq.rdata[j][i] =
            prefixData[j - 1] - prefixData[i];
        }
    }

    return sq;
}

// Sqaure root Decompostion at level3
vector<vector<sqrtD>> tripleDecompose(sqrtD sq1,
                      sqrtD sq2,vector<int> &arr)
{
    vector<vector<sqrtD>> sq(sq1.blocks,
                      vector<sqrtD>(sq1.blocks));

    int blk_idx1 = -1;

    for(int i = 0; i < sq1.blocks; i++)
    {
        int blk_ldx1 = blk_idx1 + 1;
        blk_idx1 = (i + 1) * sq1.blk_sz - 1;
        blk_idx1 = min(blk_idx1,n1 - 1);

        int blk_idx2 = blk_ldx1 - 1;

        for(int j = 0; j < sq2.blocks; ++j)
        {
            int blk_ldx2 = blk_idx2 + 1;
            blk_idx2 = blk_ldx1 + (j + 1) *
                       sq2.blk_sz - 1;
            blk_idx2 = min(blk_idx2, blk_idx1);

            vector<int> ::iterator it1 =
                        arr.begin() + blk_ldx2;
            vector<int> ::iterator it2 =
                        arr.begin() + blk_idx2 + 1;
```

```
            vector<int> vec(it1, it2);
            sq[i][j] = decompose(vec);
        }
    }
    return sq;
}

// Sqaure root Decompostion at level2
vector<sqrtD> doubleDecompose(sqrtD sq1,
                              vector<int> &arr)
{
    vector<sqrtD> sq(sq1.blocks);
    int blk_idx = -1;
    for(int i = 0; i < sq1.blocks; i++)
    {
        int blk_ldx = blk_idx + 1;
        blk_idx = (i + 1) * sq1.blk_sz - 1;
        blk_idx = min(blk_idx, n1 - 1);
        vector<int> ::iterator it1 =
                    arr.begin() + blk_ldx;
        vector<int> ::iterator it2 =
                    arr.begin() + blk_idx + 1;
        vector<int> vec(it1, it2);
        sq[i] = decompose(vec);
    }

    return sq;
}

// Sqaure root Decompostion at level1
void singleDecompose(vector<int> &arr)
{
    sqrtD sq1 = decompose(arr);
    vector<sqrtD> sq2(sq1.blocks);
    sq2 = doubleDecompose(sq1, arr);

    vector<vector<sqrtD>> sq3(sq1.blocks,
           vector<sqrtD>(sq2[0].blocks));

    sq3 = tripleDecompose(sq1, sq2[0],arr);

    // ASSIGNMENT TO GLOBAL VARIABLES
    Sq1 = sq1;
    Sq2.resize(sq1.blocks);
    Sq2 = sq2;
    Sq3.resize(sq1.blocks,
        vector<sqrtD>(sq2[0].blocks));
    Sq3 = sq3;
```

```
}

// Function for query at level 3
int queryLevel3(int start,int end, int main_blk,
                int sub_main_blk, vector<int> &arr)
{
    int blk_sz= Sq3[0][0].blk_sz;

    // Element Indexing at level2 decompostion
    int nstart = start - main_blk *
        Sq1.blk_sz - sub_main_blk * Sq2[0].blk_sz;
    int nend = end - main_blk *
        Sq1.blk_sz - sub_main_blk * Sq2[0].blk_sz;

    // Block indexing at level3 decompostion
    int st_blk = nstart / blk_sz;
    int en_blk = nend / blk_sz;

    int answer =
        Sq3[main_blk][sub_main_blk].rdata[st_blk][en_blk];

    // If start and end point dont lie in same block
    if(st_blk != en_blk)
    {
        int left = 0, en_idx = main_blk * Sq1.blk_sz +
                      sub_main_blk * Sq2[0].blk_sz +
                      (st_blk + 1) * blk_sz -1;

        for(int i = start; i <= en_idx; i++)
        {
            left += arr[i];
        }

        int right = 0, st_idx = main_blk * Sq1.blk_sz +
                       sub_main_blk * Sq2[0].blk_sz +
                       (en_blk) * blk_sz;

        for(int i = st_idx; i <= end; i++)
        {
            right += arr[i];
        }

        answer += left;
        answer += right;
    }
    else
    {
        for(int i = start; i <= end; i++)
```

```
        {
            answer += arr[i];
    }
}
return answer;
}

// Function for splitting query to level two
int queryLevel2(int start, int end, int main_blk,
                vector<int> &arr)
{
    int blk_sz = Sq2[0].blk_sz;

    // Element Indexing at level1 decompostion
    int nstart = start - (main_blk * Sq1.blk_sz);
    int nend = end - (main_blk * Sq1.blk_sz);

    // Block indexing at level2 decompostion
    int st_blk = nstart / blk_sz;
    int en_blk = nend / blk_sz;

    // Interblock data level2 decompostion
    int answer = Sq2[main_blk].rdata[st_blk][en_blk];

    if(st_blk == en_blk)
    {
        answer += queryLevel3(start, end, main_blk,
                              st_blk, arr);
    }
    else
    {
        answer += queryLevel3(start, (main_blk *
                  Sq1.blk_sz) + ((st_blk + 1) *
                  blk_sz) - 1, main_blk, st_blk, arr);

        answer += queryLevel3((main_blk * Sq1.blk_sz) +
                  (en_blk * blk_sz), end, main_blk, en_blk, arr);
    }
    return answer;
}

// Function to return answer according to query
int Query(int start,int end,vector<int>& arr)
{
    int blk_sz = Sq1.blk_sz;
    int st_blk = start / blk_sz;
    int en_blk = end / blk_sz;
```

```
    // Interblock data level1 decompostion
    int answer = Sq1.rdata[st_blk][en_blk];

    if(st_blk == en_blk)
    {
        answer += queryLevel2(start, end, st_blk, arr);
    }
    else
    {
        answer += queryLevel2(start, (st_blk + 1) *
                  blk_sz - 1, st_blk, arr);
        answer += queryLevel2(en_blk * blk_sz, end,
                              en_blk, arr);
    }

    // returning final answer
    return answer;
}

// Driver code
int main()
{
    n1 = 16;

    vector<int> arr = {7, 2, 3, 0, 5, 10, 3, 12,
                       18, 1, 2, 3, 4, 5, 6, 7};

    singleDecompose(arr);

    int q = 5;
    pair<int, int> query[q] = {{6, 10}, {7, 12},
                  {4, 13}, {4, 11}, {12, 16}};

    for(int i = 0; i < q; i++)
    {
        int a = query[i].first, b = query[i].second;
        printf("%d\n", Query(a - 1, b - 1, arr));
    }

    return 0;
}
```

Output:

```
44
39
58
```

```
51
25
```

Time Complexity : O(q * d * $n^{1/(2\hat{}3)}$) O(q * k) O(q)
Auxiliary Space : O(k * n) O(n)
Note : This article is to only explain the method of decomposing the square root to further decomposition.

Improved By : vaibhav2992

Source

https://www.geeksforgeeks.org/extended-mos-algorithm-o1-time-complexity/

Chapter 53

Fibonacci Search

Fibonacci Search - GeeksforGeeks

Given a sorted array arr[] of size n and an element x to be searched in it. Return index of x if it is present in array else return -1.
Examples:

```
Input:  arr[] = {2, 3, 4, 10, 40}, x = 10
Output:  3
Element x is present at index 3.

Input:  arr[] = {2, 3, 4, 10, 40}, x = 11
Output:  -1
Element x is not present.
```

Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.

Similarities with Binary Search:

1. Works for sorted arrays
2. A Divide and Conquer Algorithm.
3. Has Log n time complexity.

Differences with Binary Search:

1. Fibonacci Search divides given array in unequal parts
2. Binary Search uses division operator to divide range. Fibonacci Search doesn't use /, but uses + and -. The division operator may be costly on some CPUs.
3. Fibonacci Search examines relatively closer elements in subsequent steps. So when input array is big that cannot fit in CPU cache or even in RAM, Fibonacci Search can be useful.

Background:
Fibonacci Numbers are recursively defined as F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1. First few Fibinacci Numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Observations:
Below observation is used for range elimination, and hence for the O(log(n)) complexity.

```
F(n - 2) &approx; (1/3)*F(n) and
F(n - 1) &approx; (2/3)*F(n).
```

Algorithm:
Let the searched element be x.

The idea is to first find the smallest Fibonacci number that is greater than or equal to the length of given array. Let the found Fibonacci number be fib (m'th Fibonacci number). We use (m-2)'th Fibonacci number as the index (If it is a valid index). Let (m-2)'th Fibonacci Number be i, we compare arr[i] with x, if x is same, we return i. Else if x is greater, we recur for subarray after i, else we recur for subarray before i.

Below is the complete algorithm
Let arr[0..n-1] be the input array and element to be searched be x.

1. Find the smallest Fibonacci Number greater than or equal to n. Let this number be fibM [m'th Fibonacci Number]. Let the two Fibonacci numbers preceding it be fibMm1 [(m-1)'th Fibonacci Number] and fibMm2 [(m-2)'th Fibonacci Number].
2. While the array has elements to be inspected:
   (a) Compare x with the last element of the range covered by fibMm2
   (b) **If** x matches, return index
   (c) **Else If** x is less than the element, move the three Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array.
   (d) **Else** x is greater than the element, move the three Fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate elimination of approximately front one-third of the remaining array.
3. Since there might be a single element remaining for comparison, check if fibMm1 is 1. If Yes, compare x with that remaining element. If match, return index.

C

```
 // C program for Fibonacci Search
#include <stdio.h>

// Utility function to find minimum of two elements
int min(int x, int y) { return (x<=y)? x : y; }

/* Returns index of x if present,  else returns -1 */
int fibMonaccianSearch(int arr[], int x, int n)
```

```
{
    /* Initialize fibonacci numbers */
    int fibMMm2 = 0;   // (m-2)'th Fibonacci No.
    int fibMMm1 = 1;   // (m-1)'th Fibonacci No.
    int fibM = fibMMm2 + fibMMm1;  // m'th Fibonacci

    /* fibM is going to store the smallest Fibonacci
       Number greater than or equal to n */
    while (fibM < n)
    {
        fibMMm2 = fibMMm1;
        fibMMm1 = fibM;
        fibM  = fibMMm2 + fibMMm1;
    }

    // Marks the eliminated range from front
    int offset = -1;

    /* while there are elements to be inspected. Note that
       we compare arr[fibMm2] with x. When fibM becomes 1,
       fibMm2 becomes 0 */
    while (fibM > 1)
    {
        // Check if fibMm2 is a valid location
        int i = min(offset+fibMMm2, n-1);

        /* If x is greater than the value at index fibMm2,
           cut the subarray array from offset to i */
        if (arr[i] < x)
        {
            fibM  = fibMMm1;
            fibMMm1 = fibMMm2;
            fibMMm2 = fibM - fibMMm1;
            offset = i;
        }

        /* If x is greater than the value at index fibMm2,
           cut the subarray after i+1  */
        else if (arr[i] > x)
        {
            fibM  = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        }

        /* element found. return index */
        else return i;
    }
```

```
    /* comparing the last element with x */
    if(fibMMm1 && arr[offset+1]==x)return offset+1;

    /*element not found. return -1 */
    return -1;
}

/* driver function */
int main(void)
{
    int arr[] = {10, 22, 35, 40, 45, 50, 80, 82,
                 85, 90, 100};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 85;
    printf("Found at index: %d",
            fibMonaccianSearch(arr, x, n));
    return 0;
}
```

Java

```
 // Java program for Fibonacci Search
import java.util.*;

class Fibonacci
{
    // Utility function to find minimum
    // of two elements
    public static int min(int x, int y)
    { return (x <= y)? x : y; }

    /* Returns index of x if present, else returns -1 */
    public static int fibMonaccianSearch(int arr[],
                                         int x, int n)
    {
        /* Initialize fibonacci numbers */
        int fibMMm2 = 0; // (m-2)'th Fibonacci No.
        int fibMMm1 = 1; // (m-1)'th Fibonacci No.
        int fibM = fibMMm2 + fibMMm1; // m'th Fibonacci

        /* fibM is going to store the smallest
        Fibonacci Number greater than or equal to n */
        while (fibM < n)
        {
            fibMMm2 = fibMMm1;
            fibMMm1 = fibM;
            fibM = fibMMm2 + fibMMm1;
```

```
    }

    // Marks the eliminated range from front
    int offset = -1;

    /* while there are elements to be inspected.
    Note that we compare arr[fibMm2] with x.
    When fibM becomes 1, fibMm2 becomes 0 */
    while (fibM > 1)
    {
        // Check if fibMm2 is a valid location
        int i = min(offset+fibMMm2, n-1);

        /* If x is greater than the value at
        index fibMm2, cut the subarray array
        from offset to i */
        if (arr[i] < x)
        {
            fibM = fibMMm1;
            fibMMm1 = fibMMm2;
            fibMMm2 = fibM - fibMMm1;
            offset = i;
        }

        /* If x is greater than the value at index
        fibMm2, cut the subarray after i+1 */
        else if (arr[i] > x)
        {
            fibM = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        }

        /* element found. return index */
        else return i;
    }

    /* comparing the last element with x */
    if(fibMMm1 == 1 && arr[offset+1] == x)
        return offset+1;

    /*element not found. return -1 */
    return -1;
}

// driver code
public static void main(String[] args)
{
```

```
        int arr[] = {10, 22, 35, 40, 45, 50,
                     80, 82, 85, 90, 100};
        int n = 11;
        int x = 85;
        System.out.print ("Found at index: "+
                    fibMonaccianSearch(arr, x, n));
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
 # Python3 program for Fibonacci search.
from bisect import bisect_left

# Returns index of x if present,  else
# returns -1
def fibMonaccianSearch(arr, x, n):

    # Initialize fibonacci numbers
    fibMMm2 = 0 # (m-2)'th Fibonacci No.
    fibMMm1 = 1 # (m-1)'th Fibonacci No.
    fibM = fibMMm2 + fibMMm1 # m'th Fibonacci

    # fibM is going to store the smallest
    # Fibonacci Number greater than or equal to n
    while (fibM < n):
        fibMMm2 = fibMMm1
        fibMMm1 = fibM
        fibM = fibMMm2 + fibMMm1

    # Marks the eliminated range from front
    offset = -1;

    # while there are elements to be inspected.
    # Note that we compare arr[fibMm2] with x.
    # When fibM becomes 1, fibMm2 becomes 0
    while (fibM > 1):

        # Check if fibMm2 is a valid location
        i = min(offset+fibMMm2, n-1)

        # If x is greater than the value at
        # index fibMm2, cut the subarray array
        # from offset to i
        if (arr[i] < x):
            fibM = fibMMm1
```

```
            fibMMm1 = fibMMm2
            fibMMm2 = fibM - fibMMm1
            offset = i

        # If x is greater than the value at
        # index fibMm2, cut the subarray
        # after i+1
        elif (arr[i] > x):
            fibM = fibMMm2
            fibMMm1 = fibMMm1 - fibMMm2
            fibMMm2 = fibM - fibMMm1

        # element found. return index
        else :
            return i

    # comparing the last element with x */
    if(fibMMm1 and arr[offset+1] == x):
        return offset+1;

    # element not found. return -1
    return -1

# Driver Code
arr = [10, 22, 35, 40, 45, 50,
       80, 82, 85, 90, 100]
n = len(arr)
x = 85
print("Found at index:",
      fibMonaccianSearch(arr, x, n))

# This code is contributed by rishabh_jain
```

C#

```
 // C# program for Fibonacci Search
using System;

class GFG {

    // Utility function to find minimum
    // of two elements
    public static int min(int x, int y)
    {
        return (x <= y)? x : y;
    }

    /* Returns index of x if present, else returns -1 */
```

```
public static int fibMonaccianSearch(int []arr,
                                     int x, int n)
{
    /* Initialize fibonacci numbers */
    int fibMMm2 = 0; // (m-2)'th Fibonacci No.
    int fibMMm1 = 1; // (m-1)'th Fibonacci No.
    int fibM = fibMMm2 + fibMMm1; // m'th Fibonacci

    /* fibM is going to store the smallest
    Fibonacci Number greater than or equal to n */
    while (fibM < n)
    {
        fibMMm2 = fibMMm1;
        fibMMm1 = fibM;
        fibM = fibMMm2 + fibMMm1;
    }

    // Marks the eliminated range from front
    int offset = -1;

    /* while there are elements to be inspected.
    Note that we compare arr[fibMm2] with x.
    When fibM becomes 1, fibMm2 becomes 0 */
    while (fibM > 1)
    {
        // Check if fibMm2 is a valid location
        int i = min(offset+fibMMm2, n-1);

        /* If x is greater than the value at
        index fibMm2, cut the subarray array
        from offset to i */
        if (arr[i] < x)
        {
            fibM = fibMMm1;
            fibMMm1 = fibMMm2;
            fibMMm2 = fibM - fibMMm1;
            offset = i;
        }

        /* If x is greater than the value at index
        fibMm2, cut the subarray after i+1 */
        else if (arr[i] > x)
        {
            fibM = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        }
```

```
            /* element found. return index */
            else return i;
        }

        /* comparing the last element with x */
        if(fibMMm1 == 1 && arr[offset+1] == x)
            return offset + 1;

        /*element not found. return -1 */
        return -1;
    }

    // driver code
    public static void Main()
    {
        int []arr = {10, 22, 35, 40, 45, 50,
                     80, 82, 85, 90, 100};
        int n = 11;
        int x = 85;
        Console.Write ("Found at index: " +
                fibMonaccianSearch(arr, x, n));
    }
}

// This code is contributed by nitin mittal.
```

Output:

```
Found at index: 8
```

Illustration:
Let us understand the algorithm with below example:

i	1	2	3	4	5	6	7	8	9	10	11	12	13
ar[i]	**10**	**22**	**35**	**40**	**45**	**50**	**80**	**82**	**85**	**90**	**100**	-	-

Illustration assumption: 1-based indexing. Target element x is 85. Length of array n = 11.

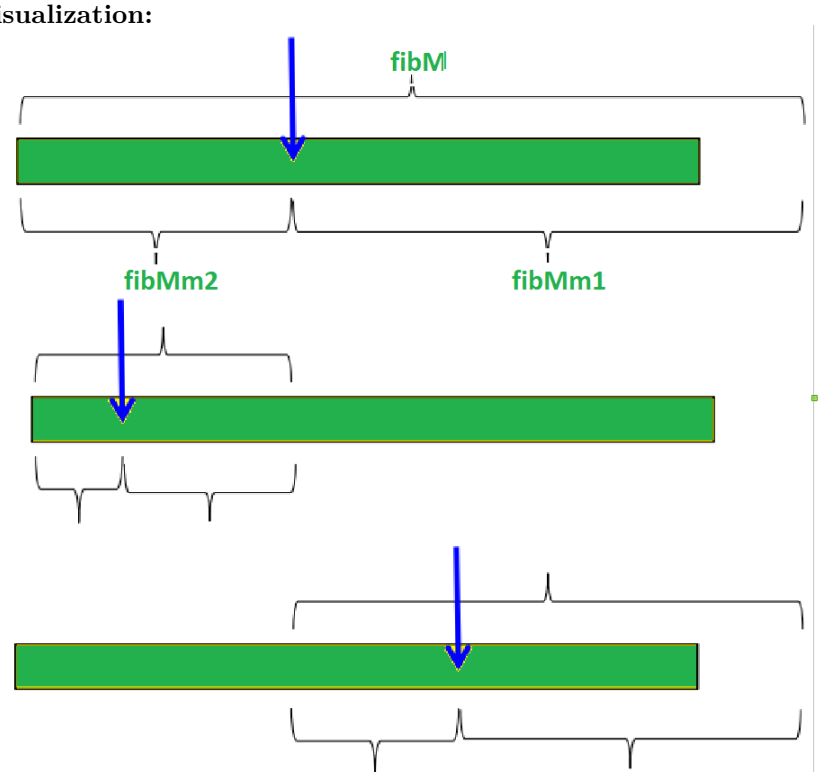
Smallest Fibonacci number greate than or equal to 11 is 13. As per our illustration, fibMm2 = 5, fibMm1 = 8, and fibM = 13.

Another implementation detail is the offset variable (zero initialized). It marks the range that has been eliminated, starting from the front. We will update it time to time.

Now since the offset value is an index and all indices including it and below it have been eliminated, it only makes sense to add something to it. Since fibMm2 marks approximately one-third of our array, as well as the indices it marks are sure to be valid ones, we can add fibMm2 to offset and check the element at index i = min(offset + fibMm2, n).

fibMm2	*fibMm1*	*fibM*	*offset*	i=min(offset+fibLn)	*arr[i]*	*Consequence*
5	8	13	0	5	45	Move one down, reset offset
3	5	8	5	8	82	Move one down, reset offset
2	3	5	8	10	90	Move two down
1	1	2	8	9	85	Return i

Visualization:



Time Complexity analysis:
The worst case will occur when we have our target in the larger (2/3) fraction of the array, as we proceed to find it. In other words, we are eliminating the smaller (1/3) fraction of the array every time. We call once for n, then for(2/3) n, then for (4/9) n and henceforth.

Consider that:

$$fib(n) = \left[\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n \right] \sim c*1.62^n$$

$$for\ n \sim c*1.62^n\ we\ make\ O(n')\ comparisons.\ We,\ thus,\ need\ O(\log(n))\ comparisons.$$

References:
https://en.wikipedia.org/wiki/Fibonacci_search_technique

This article is contributed by **Yash Varyani**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : ScottSaint, nitin mittal

Source

https://www.geeksforgeeks.org/fibonacci-search/

Chapter 54

Find a Fixed Point (Value equal to index) in a given array

Find a Fixed Point (Value equal to index) in a given array - GeeksforGeeks

Given an array of n distinct integers sorted in ascending order, write a function that returns a Fixed Point in the array, if there is any Fixed Point present in array, else returns -1. Fixed Point in an array is an index i such that arr[i] is equal to i. Note that integers in array can be negative.

Examples:

```
Input: arr[] = {-10, -5, 0, 3, 7}
Output: 3  // arr[3] == 3

Input: arr[] = {0, 2, 5, 8, 17}
Output: 0  // arr[0] == 0


Input: arr[] = {-10, -5, 3, 4, 7, 9}
Output: -1  // No Fixed Point
```

Method 1 (Linear Search)
Linearly search for an index i such that arr[i] == i. Return the first such index found. Thanks to pm for suggesting this solution.

C/C++

```
 // C/C++ program to check fixed point
// in an array using linear search
#include<stdio.h>
```

```
int linearSearch(int arr[], int n)
{
    int i;
    for(i = 0; i < n; i++)
    {
        if(arr[i] == i)
            return i;
    }

    /* If no fixed point present then return -1 */
    return -1;
}

/* Driver program to check above functions */
int main()
{
    int arr[] = {-10, -1, 0, 3, 10, 11, 30, 50, 100};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Fixed Point is %d", linearSearch(arr, n));
    getchar();
    return 0;
}
```

Java

```
 // Java program to check fixed point
// in an array using linear search

class Main
{
    static int linearSearch(int arr[], int n)
    {
        int i;
        for(i = 0; i < n; i++)
        {
            if(arr[i] == i)
                return i;
        }

        /* If no fixed point present
           then return -1 */
        return -1;
    }
    //main function
    public static void main(String args[])
    {
        int arr[] = {-10, -1, 0, 3, 10, 11, 30, 50, 100};
```

```
        int n = arr.length;
        System.out.println("Fixed Point is "
                     + linearSearch(arr, n));
    }
}
```

Python

```
 # Python program to check fixed point
# in an array using linear search
def linearSearch(arr, n):
    for i in range(n):
        if arr[i] is i:
            return i
    # If no fixed point present then return -1
    return -1

# Driver program to check above functions
arr = [-10, -1, 0, 3, 10, 11, 30, 50, 100]
n = len(arr)
print("Fixed Point is " + str(linearSearch(arr,n)))

# This code is contributed by Pratik Chhajer
```

C#

```
 // C# program to check fixed point
// in an array using linear search
using System;

class GFG
{
    static int linearSearch(int []arr, int n)
    {
        int i;
        for(i = 0; i < n; i++)
        {
            if(arr[i] == i)
                return i;
        }

        /* If no fixed point present
        then return -1 */
        return -1;
    }
    // Driver code
    public static void Main()
```

```
    {
        int []arr = {-10, -1, 0, 3, 10, 11, 30, 50, 100};
        int n = arr.Length;
        Console.Write("Fixed Point is "+ linearSearch(arr, n));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to check fixed point
// in an array using linear search

function linearSearch($arr, $n)
{
    for($i = 0; $i < $n; $i++)
    {
        if($arr[$i] == $i)
            return $i;
    }

    // If no fixed point present then
    // return -1
    return -1;
}

    // Driver Code
    $arr = array(-10, -1, 0, 3, 10,
                 11, 30, 50, 100);
    $n = count($arr);
    echo "Fixed Point is ".
            linearSearch($arr,$n);

// This code is contributed by Sam007
?>
```

Output:

```
Fixed Point is 3
```

Time Complexity: O(n)

Method 2 (Binary Search)
First check whether middle element is Fixed Point or not. If it is, then return it; otherwise check whether index of middle element is greater than value at the index. If index is greater, then Fixed Point(s) lies on the right side of the middle point (obviously only if there is a Fixed Point). Else the Fixed Point(s) lies on left side.

C/C++

```
 // C/C++ program to check fixed point
// in an array using binary search
#include<stdio.h>

int binarySearch(int arr[], int low, int high)
{
    if(high >= low)
    {
        int mid = (low + high)/2;  /*low + (high - low)/2;*/
        if(mid == arr[mid])
            return mid;
        if(mid > arr[mid])
            return binarySearch(arr, (mid + 1), high);
        else
            return binarySearch(arr, low, (mid -1));
    }

    /* Return -1 if there is no Fixed Point */
    return -1;
}

/* Driver program to check above functions */
int main()
{
    int arr[10] = {-10, -1, 0, 3, 10, 11, 30, 50, 100};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Fixed Point is %d", binarySearch(arr, 0, n-1));
    getchar();
    return 0;
}
```

Java

```
 // Java program to check fixed point
// in an array using binary search

class Main
{
    static int binarySearch(int arr[], int low, int high)
    {
```

```
        if(high >= low)
        {
            /* low + (high - low)/2; */
            int mid = (low + high)/2;
            if(mid == arr[mid])
                return mid;
            if(mid > arr[mid])
                return binarySearch(arr, (mid + 1), high);
            else
                return binarySearch(arr, low, (mid -1));
        }

        /* Return -1 if there is
           no Fixed Point */
        return -1;
    }

    //main function
    public static void main(String args[])
    {
        int arr[] = {-10, -1, 0, 3 , 10, 11, 30, 50, 100};
        int n = arr.length;
        System.out.println("Fixed Point is "
                    + binarySearch(arr,0, n-1));
    }
}
```

Python

```
 # Python program to check fixed point
# in an array using binary search
def binarySearch(arr, low, high):
    if high >= low:
        mid = (low + high)//2

    if mid is arr[mid]:
        return mid

    if mid > arr[mid]:
        return binarySearch(arr, (mid + 1), high)
    else:
        return binarySearch(arr, low, (mid -1))

    # Return -1 if there is no Fixed Point
    return -1


# Driver program to check above functions */
```

```
arr = [-10, -1, 0, 3, 10, 11, 30, 50, 100]
n = len(arr)
print("Fixed Point is " + str(binarySearch(arr, 0, n-1)))


# This code is contributed by Pratik Chhajer
```

C#

```
 // C# program to check fixed point
// in an array using binary search
using System;

class GFG
{
    static int binarySearch(int []arr, int low, int high)
    {
        if(high >= low)
        {
            // low + (high - low)/2;
            int mid = (low + high)/2;

            if(mid == arr[mid])
                return mid;
            if(mid > arr[mid])
                return binarySearch(arr, (mid + 1), high);
            else
                return binarySearch(arr, low, (mid -1));
        }

        /* Return -1 if there is
        no Fixed Point */
        return -1;
    }

    // Driver code
    public static void Main()
    {
        int []arr = {-10, -1, 0, 3 , 10, 11, 30, 50, 100};
        int n = arr.Length;
        Console.Write("Fixed Point is "+ binarySearch(arr,0, n-1));
    }
}
// This code is contributed by Sam007
```

PHP

```
 <?php
```

```
// PHP program to check fixed po
// in an array using binary search

function binarySearch($arr, $low, $high)
{
    if($high >= $low)
    {

         /*low + (high - low)/2;*/
        $mid = (int)(($low + $high) / 2);
        if($mid == $arr[$mid])
            return $mid;
        if($mid > $arr[$mid])
            return binarySearch($arr, ($mid + 1), $high);
        else
            return binarySearch($arr, $low, ($mid - 1));
    }

    /* Return -1 if there is
       no Fixed Po*/
    return -1;
}

    // Driver Code
    $arr = array(-10, -1, 0, 3, 10,
                  11, 30, 50, 100);
    $n = count($arr);
    echo "Fixed Point is: "
        . binarySearch($arr, 0, $n - 1);

// This code is contributed by Anuj_67
?>
```

Output:

```
Fixed Point is 3
```

Algorithmic Paradigm: Divide & Conquer
Time Complexity: O(Logn)

Find a Fixed Point (Value equal to index) in a given array | Duplicates Allowed

Improved By : Sam007, vt_m

Source

https://www.geeksforgeeks.org/find-a-fixed-point-in-a-given-array/

Chapter 55

Find a pair with maximum product in array of Integers

Find a pair with maximum product in array of Integers - GeeksforGeeks

Given an array with both +ive and -ive integers, return a pair with highest product.

Examples :

```
Input: arr[] = {1, 4, 3, 6, 7, 0}
Output: {6,7}

Input: arr[] = {-1, -3, -4, 2, 0, -5}
Output: {-4,-5}
```

A **Simple Solution** is to consider every pair and keep track maximum product. Below is the implementation of this simple solution.

C++

```
 // A simple C++ program to find max product pair in
// an array of integers
#include<bits/stdc++.h>
using namespace std;

// Function to find maximum product pair in arr[0..n-1]
void maxProduct(int arr[], int n)
{
    if (n < 2)
    {
        cout << "No pairs exists\n";
```

```
        return;
    }

    // Initialize max product pair
    int a = arr[0], b = arr[1];

    // Traverse through every possible pair
    // and keep track of max product
    for (int i=0; i<n; i++)
      for (int j=i+1; j<n; j++)
         if (arr[i]*arr[j] > a*b)
            a = arr[i], b = arr[j];

    cout << "Max product pair is {" << a << ", "
         << b << "}";
}

// Driver program to test
int main()
{
    int arr[] = {1, 4, 3, 6, 7, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    maxProduct(arr, n);
    return 0;
}
```

Java

```
 // JAVA Code to Find a pair with maximum
// product in array of Integers
import java.util.*;

class GFG {

    // Function to find maximum product pair
    // in arr[0..n-1]
    static void maxProduct(int arr[], int n)
    {
        if (n < 2)
        {
            System.out.println("No pairs exists");
            return;
        }

        // Initialize max product pair
        int a = arr[0], b = arr[1];

        // Traverse through every possible pair
```

```
        // and keep track of max product
        for (int i = 0; i < n; i++)
          for (int j = i + 1; j < n; j++)
             if (arr[i] * arr[j] > a * b){
                a = arr[i];
                b = arr[j];
             }

        System.out.println("Max product pair is {" +
                           a + ", " + b + "}");
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int arr[] = {1, 4, 3, 6, 7, 0};
        int n = arr.length;
        maxProduct(arr, n);

    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # A simple Python3 program to find max
# product pair in an array of integers

# Function to find maximum
# product pair in arr[0..n-1]
def maxProduct(arr, n):

    if (n < 2):
        print("No pairs exists")
        return

    # Initialize max product pair
    a = arr[0]; b = arr[1]

    # Traverse through every possible pair
    # and keep track of max product
    for i in range(0, n):

        for j in range(i + 1, n):
            if (arr[i] * arr[j] > a * b):
                a = arr[i]; b = arr[j]
```

```
    print("Max product pair is {", a, ",", b, "}",
                                          sep = "")

# Driver Code
arr = [1, 4, 3, 6, 7, 0]
n = len(arr)
maxProduct(arr, n)

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# Code to Find a pair with maximum
// product in array of Integers
using System;

class GFG
{

    // Function to find maximum
    // product pair in arr[0..n-1]
    static void maxProduct(int []arr, int n)
    {
        if (n < 2)
        {
            Console.Write("No pairs exists");
            return;
        }

        // Initialize max product pair
        int a = arr[0], b = arr[1];

        // Traverse through every possible pair
        // and keep track of max product
        for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] * arr[j] > a * b)
            {
                a = arr[i];
                b = arr[j];
            }

    Console.Write("Max product pair is {" +
                      a + ", " + b + "}");
    }

    // Driver Code
    public static void Main()
```

```
    {
        int []arr = {1, 4, 3, 6, 7, 0};
        int n = arr.Length;
        maxProduct(arr, n);
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// A simple PHP program to
// find max product pair in
// an array of integers

// Function to find maximum
// product pair in arr[0..n-1]
function maxProduct( $arr, $n)
{
    if ($n < 2)
    {
        echo "No pairs exists\n";
        return;
    }

    // Initialize max product pair
    $a = $arr[0];
    $b = $arr[1];

    // Traverse through every possible pair
    // and keep track of max product
    for ($i = 0; $i < $n; $i++)
    for ($j = $i + 1; $j < $n; $j++)
    {
        if ($arr[$i] * $arr[$j] > $a * $b)
        {
            $a = $arr[$i];
            $b = $arr[$j];
        }
    }

    echo "Max product pair is {" , $a , ", ";
    echo $b , "}";
}

    // Driver Code
    $arr = array(1, 4, 3, 6, 7, 0);
```

```
    $n = count($arr);
    maxProduct($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
Max product pair is {6, 7}
```

Time Complexity : $O(n^2)$

A **Better Solution** is to use sorting. Below are detailed steps.
1) Sort input array in increasing order.
2) If all elements are positive, then return product of last two numbers.
3) Else return maximum of products of first two and last two numbers.
Time complexity of this solution is O(nLog n). Thanks to Rahul Jain for suggesting this method.

An **Efficient Solution** can solve the above problem in single traversal of input array. The idea is to traverse the input array and keep track of following four values.
a) Maximum positive value
b) Second maximum positive value
c) Maximum negative value i.e., a negative value with maximum absolute value
d) Second maximum negative value.
At the end of the loop, compare the products of first two and last two and print the maximum of two products. Below is the implementation of this idea.

C++

```
 // A O(n) C++ program to find maximum product pair in an array
#include<bits/stdc++.h>
using namespace std;

// Function to find maximum product pair in arr[0..n-1]
void maxProduct(int arr[], int n)
{
    if (n < 2)
    {
        cout << "No pairs exists\n";
        return;
    }

    if (n == 2)
    {
        cout << arr[0] << " " << arr[1] << endl;
```

```
        return;
    }

    // Iniitialize maximum and second maximum
    int posa = INT_MIN, posb = INT_MIN;

    // Iniitialize minimum and second minimum
    int nega = INT_MIN, negb = INT_MIN;

    // Traverse given array
    for (int i = 0; i < n; i++)
    {
        // Update maximum and second maximum if needed
        if (arr[i] > posa)
        {
            posb = posa;
            posa = arr[i];
        }
        else if (arr[i] > posb)
            posb = arr[i];

        // Update minimum and second minimum if needed
        if (arr[i] < 0 && abs(arr[i]) > abs(nega))
        {
            negb = nega;
            nega = arr[i];
        }
        else if(arr[i] < 0 && abs(arr[i]) > abs(negb))
            negb = arr[i];
    }

    if (nega*negb > posa*posb)
        cout << "Max product pair is {" << nega << ", "
             << negb << "}";
    else
        cout << "Max product pair is {" << posa << ", "
             << posb << "}";
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 4, 3, 6, 7, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    maxProduct(arr, n);
    return 0;
}
```

Java

```
 // JAVA Code to Find a pair with maximum
// product in array of Integers
import java.util.*;

class GFG {

    // Function to find maximum product pair
    // in arr[0..n-1]
    static void maxProduct(int arr[], int n)
    {
        if (n < 2)
        {
            System.out.println("No pairs exists");
            return;
        }

        if (n == 2)
        {
            System.out.println(arr[0] + " " + arr[1]);
            return;
        }

        // Iniitialize maximum and second maximum
        int posa = Integer.MIN_VALUE,
            posb = Integer.MIN_VALUE;

        // Iniitialize minimum and second minimum
        int nega = Integer.MIN_VALUE,
            negb = Integer.MIN_VALUE;

        // Traverse given array
        for (int i = 0; i < n; i++)
        {
            // Update maximum and second maximum
            // if needed
            if (arr[i] > posa)
            {
                posb = posa;
                posa = arr[i];
            }
            else if (arr[i] > posb)
                posb = arr[i];

            // Update minimum and second minimum
            // if needed
            if (arr[i] < 0 && Math.abs(arr[i]) >
```

```
                    Math.abs(nega))
            {
                negb = nega;
                nega = arr[i];
            }
            else if(arr[i] < 0 && Math.abs(arr[i])
                    > Math.abs(negb))
                negb = arr[i];
        }

        if (nega * negb > posa * posb)
            System.out.println("Max product pair is {"
                        + nega + ", " + negb + "}");
        else
            System.out.println("Max product pair is {"
                        + posa + ", " + posb + "}");
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int arr[] = {1, 4, 3, 6, 7, 0};
        int n = arr.length;
        maxProduct(arr, n);

    }
}

// This code is contributed by Arnav Kr. Mandal.
```

C#

```
 // C# Code to Find a pair with maximum
// product in array of Integers
using System;
class GFG {

    // Function to find maximum
    // product pair in arr[0..n-1]
    static void maxProduct(int []arr, int n)
    {
        if (n < 2)
        {
            Console.WriteLine("No pairs exists");
            return;
        }

        if (n == 2)
```

```
{
    Console.WriteLine(arr[0] + " " + arr[1]);
    return;
}

// Iniitialize maximum and
// second maximum
int posa = int.MinValue;
int posb = int.MaxValue;

// Iniitialize minimum and
// second minimum
int nega = int.MinValue;
int negb = int.MaxValue;

// Traverse given array
for (int i = 0; i < n; i++)
{

    // Update maximum and
    // second maximum
    // if needed
    if (arr[i] > posa)
    {
        posb = posa;
        posa = arr[i];
    }
    else if (arr[i] > posb)
        posb = arr[i];

    // Update minimum and
    // second minimum if
    // needed
    if (arr[i] < 0 && Math.Abs(arr[i]) >
                      Math.Abs(nega))
    {
        negb = nega;
        nega = arr[i];
    }
    else if(arr[i] < 0 &&
            Math.Abs(arr[i]) >
            Math.Abs(negb))
        negb = arr[i];
}

if (nega * negb > posa * posb)
    Console.WriteLine("Max product pair is {"
                + nega + ", " + negb + "}");
```

```
        else
            Console.WriteLine("Max product pair is {"
                        + posa + ", " + posb + "}");
    }

    // Driver Code
    public static void Main()
    {
        int []arr = {1, 4, 3, 6, 7, 0};
        int n = arr.Length;
        maxProduct(arr, n);
    }
}

// This code is contributed by anuj_67.
```

Output:

```
Max product pair is {7, 6}
```

Time complexity: O(n)
Auxiliary Space: O(1)

Thanks to Gaurav Ahirwar for suggesting this method.

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/return-a-pair-with-maximum-product-in-array-of-integers/

Chapter 56

Find a pair with the given difference

Find a pair with the given difference - GeeksforGeeks

Given an unsorted array and a number n, find if there exists a pair of elements in the array whose difference is n.

```
Examples:
Input: arr[] = {5, 20, 3, 2, 50, 80}, n = 78
Output: Pair Found: (2, 80)

Input: arr[] = {90, 70, 20, 80, 50}, n = 45
Output: No Such Pair
```

The simplest method is to run two loops, the outer loop picks the first element (smaller element) and the inner loop looks for the element picked by outer loop plus n. Time complexity of this method is O(n^2).

We can use sorting and Binary Search to improve time complexity to O(nLogn). The first step is to sort the array in ascending order. Once the array is sorted, traverse the array from left to right, and for each element arr[i], binary search for arr[i] + n in arr[i+1..n-1]. If the element is found, return the pair.
Both first and second steps take O(nLogn). So overall complexity is O(nLogn).

The second step of the above algorithm can be improved to O(n). The first step remain same. The idea for second step is take two index variables i and j, initialize them as 0 and 1 respectively. Now run a linear loop. If arr[j] – arr[i] is smaller than n, we need to look for greater arr[j], so increment j. If arr[j] – arr[i] is greater than n, we need to look for greater arr[i], so increment i. Thanks to Aashish Barnwal for suggesting this approach.
The following code is only for the second step of the algorithm, it assumes that the array is already sorted.

C/C++

```
 // C/C++ program to find a pair with the given difference
#include <stdio.h>

// The function assumes that the array is sorted
bool findPair(int arr[], int size, int n)
{
    // Initialize positions of two elements
    int i = 0;
    int j = 1;

    // Search for a pair
    while (i<size && j<size)
    {
        if (i != j && arr[j]-arr[i] == n)
        {
            printf("Pair Found: (%d, %d)", arr[i], arr[j]);
            return true;
        }
        else if (arr[j]-arr[i] < n)
            j++;
        else
            i++;
    }

    printf("No such pair");
    return false;
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 8, 30, 40, 100};
    int size = sizeof(arr)/sizeof(arr[0]);
    int n = 60;
    findPair(arr, size, n);
    return 0;
}
```

Java

```
 // Java program to find a pair with the given difference
import java.io.*;

class PairDifference
{
    // The function assumes that the array is sorted
    static boolean findPair(int arr[],int n)
    {
```

```
        int size = arr.length;

        // Initialize positions of two elements
        int i = 0, j = 1;

        // Search for a pair
        while (i < size && j < size)
        {
            if (i != j && arr[j]-arr[i] == n)
            {
                System.out.print("Pair Found: "+
                                 "( "+arr[i]+", "+ arr[j]+" )");
                return true;
            }
            else if (arr[j] - arr[i] < n)
                j++;
            else
                i++;
        }

        System.out.print("No such pair");
        return false;
    }

    // Driver program to test above function
    public static void main (String[] args)
    {
        int arr[] = {1, 8, 30, 40, 100};
        int n = 60;
        findPair(arr,n);
    }
}
/*This code is contributed by Devesh Agrawal*/
```

Python

```
 # Python program to find a pair with the given difference

# The function assumes that the array is sorted
def findPair(arr,n):

    size = len(arr)

    # Initialize positions of two elements
    i,j = 0,1

    # Search for a pair
    while i < size and j < size:
```

```
        if i != j and arr[j]-arr[i] == n:
            print "Pair found (",arr[i],",",arr[j],")"
            return True

        elif arr[j] - arr[i] < n:
            j+=1
        else:
            i+=1
    print "No pair found"
    return False

# Driver function to test above function
arr = [1, 8, 30, 40, 100]
n = 60
findPair(arr, n)

# This code is contributed by Devesh Agrawal
```

C#

```
 // C# program to find a pair with the given difference
using System;

class GFG {

    // The function assumes that the array is sorted
    static bool findPair(int []arr, int n)
    {
        int size = arr.Length;

        // Initialize positions of two elements
        int i = 0, j = 1;

        // Search for a pair
        while (i < size && j < size)
        {
            if (i != j && arr[j] - arr[i] == n)
            {
                Console.Write("Pair Found: "
                + "( " + arr[i] + ", " + arr[j] +" )");

                return true;
            }
            else if (arr[j] - arr[i] < n)
                j++;
            else
                i++;
```

```
        }

        Console.Write("No such pair");

        return false;
    }

    // Driver program to test above function
    public static void Main ()
    {
        int []arr = {1, 8, 30, 40, 100};
        int n = 60;

        findPair(arr, n);
    }
}

// This code is contributed by Sam007.
```

Output:

```
Pair Found: (40, 100)
```

Hashing can also be used to solve this problem. Create an empty hash table HT. Traverse the array, use array elements as hash keys and enter them in HT. Traverse the array again look for value n + arr[i] in HT.

Source

https://www.geeksforgeeks.org/find-a-pair-with-the-given-difference/

Chapter 57

Find a peak element

Find a peak element - GeeksforGeeks

Given an array of integers. Find a peak element in it. An array element is peak if it is NOT smaller than its neighbors. For corner elements, we need to consider only one neighbor. For example, for input array {5, 10, 20, 15}, 20 is the only peak element. For input array {10, 20, 15, 2, 23, 90, 67}, there are two peak elements: 20 and 90. Note that we need to return any one peak element.

Following corner cases give better idea about the problem.
1) If input array is sorted in strictly increasing order, the last element is always a peak element. For example, 50 is peak element in {10, 20, 30, 40, 50}.
2) If input array is sorted in strictly decreasing order, the first element is always a peak element. 100 is the peak element in {100, 80, 60, 50, 20}.
3) If all elements of input array are same, every element is a peak element.

It is clear from above examples that there is always a peak element in input array in any input array.

A **simple solution** is to do a linear scan of array and as soon as we find a peak element, we return it. The worst case time complexity of this method would be O(n).

Can we find a peak element in worst time complexity better than O(n)?
We can use Divide and Conquerto find a peak in O(Logn) time. The idea is Binary Search based, we compare middle element with its neighbors. If middle element is not smaller than any of its neighbors, then we return it. If the middle element is smaller than the its left neighbor, then there is always a peak in left half (Why? take few examples). If the middle element is smaller than the its right neighbor, then there is always a peak in right half (due to same reason as left half). Following are C and Java implementations of this approach.

C/C++

```
 // A C++ program to find a peak element element using divide and conquer
#include <stdio.h>

// A binary search based function that returns index of a peak element
```

```
int findPeakUtil(int arr[], int low, int high, int n)
{
    // Find index of middle element
    int mid = low + (high - low)/2;  /* (low + high)/2 */

    // Compare middle element with its neighbours (if neighbours exist)
    if ((mid == 0 || arr[mid-1] <= arr[mid]) &&
            (mid == n-1 || arr[mid+1] <= arr[mid]))
        return mid;

    // If middle element is not peak and its left neighbour is greater
    // than it, then left half must have a peak element
    else if (mid > 0 && arr[mid-1] > arr[mid])
        return findPeakUtil(arr, low, (mid -1), n);

    // If middle element is not peak and its right neighbour is greater
    // than it, then right half must have a peak element
    else return findPeakUtil(arr, (mid + 1), high, n);
}

// A wrapper over recursive function findPeakUtil()
int findPeak(int arr[], int n)
{
    return findPeakUtil(arr, 0, n-1, n);
}

/* Driver program to check above functions */
int main()
{
    int arr[] = {1, 3, 20, 4, 1, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Index of a peak point is %d", findPeak(arr, n));
    return 0;
}
```

Java

```
 // A Java program to find a peak element element using divide and conquer
import java.util.*;
import java.lang.*;
import java.io.*;

class PeakElement
{
    // A binary search based function that returns index of a peak
    // element
    static int findPeakUtil(int arr[], int low, int high, int n)
    {
```

```
        // Find index of middle element
        int mid = low + (high - low)/2;  /* (low + high)/2 */

        // Compare middle element with its neighbours (if neighbours
        // exist)
        if ((mid == 0 || arr[mid-1] <= arr[mid]) && (mid == n-1 ||
             arr[mid+1] <= arr[mid]))
            return mid;

        // If middle element is not peak and its left neighbor is
        // greater than it,then left half must have a peak element
        else if (mid > 0 && arr[mid-1] > arr[mid])
            return findPeakUtil(arr, low, (mid -1), n);

        // If middle element is not peak and its right neighbor
        // is greater than it, then right half must have a peak
        // element
        else return findPeakUtil(arr, (mid + 1), high, n);
    }

    // A wrapper over recursive function findPeakUtil()
    static int findPeak(int arr[], int n)
    {
        return findPeakUtil(arr, 0, n-1, n);
    }

    // Driver method
    public static void main (String[] args)
    {
        int arr[] = {1, 3, 20, 4, 1, 0};
        int n = arr.length;
        System.out.println("Index of a peak point is " +
                           findPeak(arr, n));
    }
}
```

Python3

```
 # A python 3 program to find a peak
# element element using divide and conquer

# A binary search based function
# that returns index of a peak element
def findPeakUtil(arr, low, high, n):

     # Find index of middle element
     # (low + high)/2
     mid = low + (high - low)/2
```

```
     mid = int(mid)

    # Compare middle element with its
    # neighbours (if neighbours exist)
    if ((mid == 0 or arr[mid - 1] <= arr[mid]) and
       (mid == n - 1 or arr[mid + 1] <= arr[mid])):
        return mid


    # If middle element is not peak and
    # its left neighbour is greater
    # than it, then left half must
    # have a peak element
    elif (mid > 0 and arr[mid - 1] > arr[mid]):
        return findPeakUtil(arr, low, (mid - 1), n)

    # If middle element is not peak and
    # its right neighbour is greater
    # than it, then right half must
    # have a peak element
    else:
        return findPeakUtil(arr, (mid + 1), high, n)


# A wrapper over recursive
# function findPeakUtil()
def findPeak(arr, n):

    return findPeakUtil(arr, 0, n - 1, n)


# Driver code
arr = [1, 3, 20, 4, 1, 0]
n = len(arr)
print("Index of a peak point is", findPeak(arr, n))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // A C# program to find
// a peak element element
// using divide and conquer
using System;

class GFG
{
```

```
// A binary search based
// function that returns
// index of a peak element
static int findPeakUtil(int []arr, int low,
                        int high, int n)
{
    // Find index of
    // middle element
    int mid = low + (high - low) / 2;

    // Compare middle element with
    // its neighbours (if neighbours
    // exist)
    if ((mid == 0 ||
         arr[mid - 1] <= arr[mid]) &&
        (mid == n - 1 ||
         arr[mid + 1] <= arr[mid]))
        return mid;

    // If middle element is not
    // peak and its left neighbor
    // is greater than it,then
    // left half must have a
    // peak element
    else if (mid > 0 &&
             arr[mid - 1] > arr[mid])
        return findPeakUtil(arr, low,
                            (mid - 1), n);

    // If middle element is not
    // peak and its right neighbor
    // is greater than it, then
    // right half must have a peak
    // element
    else return findPeakUtil(arr, (mid + 1),
                             high, n);
}

// A wrapper over recursive
// function findPeakUtil()
static int findPeak(int []arr,
                    int n)
{
    return findPeakUtil(arr, 0,
                        n - 1, n);
}
```

```
    // Driver Code
    static public void Main ()
    {
        int []arr = {1, 3, 20,
                     4, 1, 0};
        int n = arr.Length;
        Console.WriteLine("Index of a peak " +
                                  "point is " +
                            findPeak(arr, n));
    }
}

// This code is contributed by ajit
```

PHP

```
 <?php
// A PHP program to find a
// peak element element using
// divide and conquer

// A binary search based function
// that returns index of a peak
// element
function findPeakUtil($arr, $low,
                      $high, $n)
{
    // Find index of middle element
    $mid = $low + ($high - $low) / 2; // (low + high)/2

    // Compare middle element with
    // its neighbours (if neighbours exist)
    if (($mid == 0 ||
         $arr[$mid - 1] <= $arr[$mid]) &&
        ($mid == $n - 1 ||
         $arr[$mid + 1] <= $arr[$mid]))
        return $mid;

    // If middle element is not peak
    // and its left neighbour is greater
    // than it, then left half must
    // have a peak element
    else if ($mid > 0 &&
             $arr[$mid - 1] > $arr[$mid])
        return findPeakUtil($arr, $low,
                            ($mid - 1), $n);

    // If middle element is not peak
```

```
    // and its right neighbour is
    // greater than it, then right
    // half must have a peak element
    else return(findPeakUtil($arr, ($mid + 1),
                             $high, $n));
}

// A wrapper over recursive
// function findPeakUtil()
function findPeak($arr, $n)
{
    return floor(findPeakUtil($arr, 0,
                              $n - 1, $n));
}

// Driver Code
$arr = array(1, 3, 20, 4, 1, 0);
$n = sizeof($arr);
echo "Index of a peak point is ",
              findPeak($arr, $n);

// This code is contributed by ajit
?>
```

Output :

```
Index of a peak point is 2
```

Time Complexity: O(Logn) where n is number of elements in input array.

Exercise:
Consider the following modified definition of peak element. An array element is peak if it is greater than its neighbors. Note that an array may not contain a peak element with this modified definition.

References:
http://courses.csail.mit.edu/6.006/spring11/lectures/lec02.pdf
http://www.youtube.com/watch?v=HtSuA80QTyo

Related Problem:
Find local minima in an array

Improved By : jit_t

Source

https://www.geeksforgeeks.org/find-a-peak-in-a-given-array/

Chapter 58

Find a triplet such that sum of two equals to third element

Find a triplet such that sum of two equals to third element - GeeksforGeeks

Given an array of integers you have to find three numbers such that sum of two elements equals the third element.

Examples:

```
Input : {5, 32, 1, 7, 10, 50, 19, 21, 2}
Output : 21, 2, 19

Input : {5, 32, 1, 7, 10, 50, 19, 21, 0}
Output : no such triplet exist
```

Question source : Arcesium Interview Experience | Set 7 (On campus for Internship)

Simple approach: Run three loops and check if there exists a triplet such that sum of two elements equals the third element.
Time complexity : O(n^3)

Efficient approach : The idea is similar to Find a triplet that sum to a given value.

Sort the array first.

Start fixing the greatest element of three from back and traverse the array to find other two numbers which sum upto the third element.

Time complexity : O(n^2)

C++

```
 // CPP program to find three numbers
// such that sum of two makes the
```

```
// third element in array
#include <bits/stdc++.h>
using namespace std;

// utility function for finding
// triplet in array
void findTriplet(int arr[], int n)
{
    // sort the array
    sort(arr, arr + n);

    // for every element in arr
    // check if a pair exist(in array) whose
    // sum is equal to arr element
    for (int i = n - 1; i >= 0; i--) {
        int j = 0;
        int k = i - 1;
        while (j < k) {
            if (arr[i] == arr[j] + arr[k]) {

                // pair found
                cout << "numbers are " << arr[i] << " "
                     << arr[j] << " " << arr[k] << endl;
                return;
            } else if (arr[i] > arr[j] + arr[k])
                j += 1;
            else
                k -= 1;
        }
    }

    // no such triplet is found in array
    cout << "No such triplet exists";
}

// driver program
int main()
{
    int arr[] = { 5, 32, 1, 7, 10, 50, 19, 21, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    findTriplet(arr, n);
    return 0;
}
```

Java

```
 // Java program to find three numbers
```

```
// such that sum of two makes the
// third element in array
import java.util.Arrays;

public class GFG {

    // utility function for finding
    // triplet in array
    static void findTriplet(int arr[], int n)
    {
        // sort the array
        Arrays.sort(arr);

        // for every element in arr
        // check if a pair exist(in array) whose
        // sum is equal to arr element
        for (int i = n - 1; i >= 0; i--) {
            int j = 0;
            int k = i - 1;
            while (j < k) {
                if (arr[i] == arr[j] + arr[k]) {

                    // pair found
                    System.out.println("numbers are " + arr[i] + " "
                            + arr[j] + " " + arr[k]);

                    return;
                } else if (arr[i] > arr[j] + arr[k])
                    j += 1;
                else
                    k -= 1;
            }
        }

        // no such triplet is found in array
        System.out.println("No such triplet exists");
    }

    // driver program
    public static void main(String args[])
    {
        int arr[] = { 5, 32, 1, 7, 10, 50, 19, 21, 2 };
        int n = arr.length;
        findTriplet(arr, n);
    }
}
// This code is contributed by Sumit Ghosh
```

Python

```
 # Python program to find three numbers
# such that sum of two makes the
# third element in array

# utility function for finding
# triplet in array
def findTriplet(arr, n):

    # sort the array
    arr.sort()

    # for every element in arr
    # check if a pair exist(in array) whose
    # sum is equal to arr element
    i = n - 1
    while(i >= 0):
        j = 0
        k = i - 1
        while (j < k):
            if (arr[i] == arr[j] + arr[k]):

                # pair found
                print "numbers are ", arr[i], arr[j], arr[k]
                return
            elif (arr[i] > arr[j] + arr[k]):
                j += 1
            else:
                k -= 1
        i -= 1

    # no such triplet is found in array
    print "No such triplet exists"

# driver program
arr = [ 5, 32, 1, 7, 10, 50, 19, 21, 2 ]
n = len(arr)
findTriplet(arr, n)

# This code is contributed by Sachin Bisht
```

C#

```
 // C# program to find three numbers
// such that sum of two makes the
// third element in array
```

```
using System;

public class GFG {

    // utility function for finding
    // triplet in array
    static void findTriplet(int []arr, int n)
    {

        // sort the array
        Array.Sort(arr);

        // for every element in arr
        // check if a pair exist(in
        // array) whose sum is equal
        // to arr element
        for (int i = n - 1; i >= 0; i--)
        {
            int j = 0;
            int k = i - 1;
            while (j < k) {
                if (arr[i] == arr[j] + arr[k])
                {

                    // pair found
                    Console.WriteLine("numbers are "
                              + arr[i] + " "+ arr[j]
                                    + " " + arr[k]);

                    return;
                }
                else if (arr[i] > arr[j] + arr[k])
                    j += 1;
                else
                    k -= 1;
            }
        }

        // no such triplet is found in array
        Console.WriteLine("No such triplet exists");
    }

    // driver program
    public static void Main()
    {
        int []arr = { 5, 32, 1, 7, 10, 50,
                                  19, 21, 2 };
        int n = arr.Length;
```

```
        findTriplet(arr, n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find three
// numbers such that sum of
// two makes the third
// element in array

// utility function for
// finding triplet in array
function findTriplet($arr, $n)
{
    // sort the array
    sort($arr);

    // for every element in
    // arr check if a pair
    // exist(in array) whose
    // sum is equal to arr element
    for ($i = $n - 1; $i >= 0; $i--)
    {
        $j = 0;
        $k = $i - 1;
        while ($j < $k)
        {
            if ($arr[$i] == $arr[$j] + $arr[$k])
            {

                // pair found
                echo "numbers are " , $arr[$i] , " " ,
                                      $arr[$j] , " " ,
                                      $arr[$k];
                return;
            }
            else if ($arr[$i] > $arr[$j] +
                                $arr[$k])
                $j += 1;
            else
                $k -= 1;
        }
    }
```

```
    // no such triplet
    // is found in array
    echo "No such triplet exists";
}

// Driver Code
$arr = array(5, 32, 1, 7, 10,
             50, 19, 21, 2 );
$n = count($arr);

findTriplet($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
numbers are 21 2 19
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/find-triplet-sum-two-equals-third-element/

Chapter 59

Find all triplets with zero sum

Find all triplets with zero sum - GeeksforGeeks

Given an array of distinct elements. The task is to find triplets in array whose sum is zero.

Examples :

```
Input : arr[] = {0, -1, 2, -3, 1}
Output : 0 -1 1
         2 -3 1

Input : arr[] = {1, -2, 1, 0, 5}
Output : 1 -2  1
```

Method 1 (Simple : $O(n^3)$)
The naive approach is that run three loops and check one by one that sum of three elements is zero or not if sum of three elements is zero then print elements other wise print not found.

C++

```
 // A simple C++ program to find three elements
// whose sum is equal to zero
#include<bits/stdc++.h>
using namespace std;

// Prints all triplets in arr[] with 0 sum
void findTriplets(int arr[], int n)
{
    bool found = true;
    for (int i=0; i<n-2; i++)
    {
```

```
        for (int j=i+1; j<n-1; j++)
        {
            for (int k=j+1; k<n; k++)
            {
                if (arr[i]+arr[j]+arr[k] == 0)
                {
                    cout << arr[i] << " "
                         << arr[j] << " "
                         << arr[k] <<endl;
                    found = true;
                }
            }
        }
    }

    // If no triplet with 0 sum found in array
    if (found == false)
        cout << " not exist "<<endl;

}

// Driver code
int main()
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    findTriplets(arr, n);
    return 0;
}
```

Java

```
 // A simple Java program to find three elements
// whose sum is equal to zero
class num{
// Prints all triplets in arr[] with 0 sum
static void findTriplets(int[] arr, int n)
{
    boolean found = true;
    for (int i=0; i<n-2; i++)
    {
        for (int j=i+1; j<n-1; j++)
        {
            for (int k=j+1; k<n; k++)
            {
                if (arr[i]+arr[j]+arr[k] == 0)
                {
                    System.out.print(arr[i]);
```

```
                    System.out.print(" ");
                    System.out.print(arr[j]);
                    System.out.print(" ");
                    System.out.print(arr[k]);
                    System.out.print("\n");
                    found = true;
                }
            }
        }
    }

    // If no triplet with 0 sum found in array
    if (found == false)
        System.out.println(" not exist ");

}

// Driver code
public static void main(String[] args)
{
    int arr[] = {0, -1, 2, -3, 1};
    int n =arr.length;
    findTriplets(arr, n);

}
}
//This code is contributed by
//Smitha Dinesh Semwal
```

Python3

```
 # A simple Python 3 program
# to find three elements whose
# sum is equal to zero

# Prints all triplets in
# arr[] with 0 sum
def findTriplets(arr, n):

    found = True
    for i in range(0, n-2):

        for j in range(i+1, n-1):

            for k in range(j+1, n):

                if (arr[i] + arr[j] + arr[k] == 0):
                    print(arr[i], arr[j], arr[k])
```

```
                    found = True


    # If no triplet with 0 sum
    # found in array
    if (found == False):
        print(" not exist ")

# Driver code
arr = [0, -1, 2, -3, 1]
n = len(arr)
findTriplets(arr, n)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // A simple C# program to find three elements
// whose sum is equal to zero
using System;

class GFG {

    // Prints all triplets in arr[] with 0 sum
    static void findTriplets(int []arr, int n)
    {
        bool found = true;
        for (int i = 0; i < n-2; i++)
        {
            for (int j = i+1; j < n-1; j++)
            {
                for (int k = j+1; k < n; k++)
                {
                    if (arr[i] + arr[j] + arr[k]
                                            == 0)
                    {
                        Console.Write(arr[i]);
                        Console.Write(" ");
                        Console.Write(arr[j]);
                        Console.Write(" ");
                        Console.Write(arr[k]);
                        Console.Write("\n");
                        found = true;
                    }
                }
            }
        }
```

```
        // If no triplet with 0 sum found in
        // array
        if (found == false)
            Console.Write(" not exist ");
    }

    // Driver code
    public static void Main()
    {
        int []arr = {0, -1, 2, -3, 1};
        int n = arr.Length;
        findTriplets(arr, n);
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// A simple PHP program to
// find three elements whose
// sum is equal to zero

// Prints all triplets
// in arr[] with 0 sum
function findTriplets($arr, $n)
{
    $found = true;
    for ($i = 0; $i < $n - 2; $i++)
    {
        for ($j = $i + 1; $j < $n - 1; $j++)
        {
            for ($k = $j + 1; $k < $n; $k++)
            {
                if ($arr[$i] + $arr[$j] +
                              $arr[$k] == 0)
                {
                    echo $arr[$i] , " ",
                         $arr[$j] , " ",
                         $arr[$k] ,"\n";
                    $found = true;
                }
            }
        }
    }

    // If no triplet with 0
```

```
    // sum found in array
    if ($found == false)
        echo " not exist ", "\n";

}

// Driver Code
$arr = array (0, -1, 2, -3, 1);
$n = sizeof($arr);
findTriplets($arr, $n);

// This code is contributed by m_kit
?>


0 -1 1
2 -3 1
```

Time Complexity : $O(n^3)$
Auxiliary Space : $O(1)$

Method 2 (Hashing : $O(n^2)$)
We iterate through every element. For every element arr[i], we find a pair with sum "-arr[i]". This problem reduces to pairs sum and can be solved in O(n) time using hashing.

```
Run a loop from i=0 to n-2
  Create an empty hash table
  Run inner loop from j=i+1 to n-1
      If -(arr[i] + arr[j]) is present in hash table
         print arr[i], arr[j] and -(arr[i]+arr[j])
      Else
         Insert arr[j] in hash table.
```

C++

```
 // C++ program to find triplets in a given
// array whose sum is zero
#include<bits/stdc++.h>
using namespace std;

// function to print triplets with 0 sum
void findTriplets(int arr[], int n)
{
    bool found = false;

    for (int i=0; i<n-1; i++)
```

```
    {
        // Find all pairs with sum equals to
        // "-arr[i]"
        unordered_set<int> s;
        for (int j=i+1; j<n; j++)
        {
            int x = -(arr[i] + arr[j]);
            if (s.find(x) != s.end())
            {
                printf("%d %d %d\n", x, arr[i], arr[j]);
                found = true;
            }
            else
                s.insert(arr[j]);
        }
    }

    if (found == false)
        cout << " No Triplet Found" << endl;
}

// Driver code
int main()
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    findTriplets(arr, n);
    return 0;
}


-1 0 1
-3 2 1
```

Time Complexity : $O(n^2)$
Auxiliary Space : $O(n)$

Method 3 (Sorting : $O(n^2)$)

The above method requires extra space. We can solve in O(1) extra space. The idea is based on method 2 of this post.

```
1. Sort all element of array
2. Run loop from i=0 to n-2.
     Initialize two index variables l=i+1 and r=n-1
4. while (l < r)
     Check sum of arr[i], arr[l], arr[r] is
```

```
     zero or not if sum is zero then print the
     triplet and do l++ and r--.
5. If sum is less than zero then l++
6. If sum is greater than zero then r--
7. If not exist in array then print not found.
```

C++

```
 // C++ program to find triplets in a given
// array whose sum is zero
#include<bits/stdc++.h>
using namespace std;

// function to print triplets with 0 sum
void findTriplets(int arr[], int n)
{
    bool found = false;

    // sort array elements
    sort(arr, arr+n);

    for (int i=0; i<n-1; i++)
    {
        // initialize left and right
        int l = i + 1;
        int r = n - 1;
        int x = arr[i];
        while (l < r)
        {
            if (x + arr[l] + arr[r] == 0)
            {
                // print elements if it's sum is zero
                printf("%d %d %d\n", x, arr[l], arr[r]);
                l++;
                r--;
                found = true;
            }

            // If sum of three elements is less
            // than zero then increment in left
            else if (x + arr[l] + arr[r] < 0)
                l++;

            // if sum is greater than zero than
            // decrement in right side
            else
                r--;
        }
```

```
    }

    if (found == false)
        cout << " No Triplet Found" << endl;
}

// Driven source
int main()
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    findTriplets(arr, n);
    return 0;
}
```

Python3

```
 # python program to find triplets in a given
# array whose sum is zero

# function to print triplets with 0 sum
def findTriplets(arr, n):

    found = False

    # sort array elements
    arr.sort()

    for i in range(0, n-1):

        # initialize left and right
        l = i + 1
        r = n - 1
        x = arr[i]
        while (l < r):

            if (x + arr[l] + arr[r] == 0):
                # print elements if it's sum is zero
                print(x, arr[l], arr[r])
                l+=1
                r-=1
                found = True


            # If sum of three elements is less
            # than zero then increment in left
            elif (x + arr[l] + arr[r] < 0):
                l+=1
```

```
            # if sum is greater than zero than
            # decrement in right side
            else:
                r-=1

    if (found == False):
        print(" No Triplet Found")


# Driven source
arr = [0, -1, 2, -3, 1]
n = len(arr)
findTriplets(arr, n)

# This code is contributed by Smitha Dinesh Semwal
```

PHP

```
 <?php
// PHP program to find
// triplets in a given
// array whose sum is zero

// function to print
// triplets with 0 sum
function findTriplets($arr, $n)
{
    $found = false;

    // sort array elements
    sort($arr);

    for ($i = 0; $i < $n - 1; $i++)
    {
        // initialize left
        // and right
        $l = $i + 1;
        $r = $n - 1;
        $x = $arr[$i];
        while ($l < $r)
        {
            if ($x + $arr[$l] +
                     $arr[$r] == 0)
            {
                // print elements if
                // it's sum is zero
                echo $x," ", $arr[$l],
```

```
                        " ", $arr[$r], "\n";
                $l++;
                $r--;
                $found = true;
            }

            // If sum of three elements
            // is less than zero then
            // increment in left
            else if ($x + $arr[$l] +
                         $arr[$r] < 0)
                $l++;

            // if sum is greater than
            // zero than decrement
            // in right side
            else
                $r--;
        }
    }

    if ($found == false)
        echo " No Triplet Found" ,"\n";
}

// Driver Code
$arr = array (0, -1, 2, -3, 1);
$n = sizeof($arr);
findTriplets($arr, $n);

// This code is contributed by ajit
?>
```

Output :

```
-3 1 2
-1 0 1
```

Time Complexity : $O(n^2)$
Auxiliary Space : $O(1)$

Improved By : nitin mittal, jit_t

Source

https://www.geeksforgeeks.org/find-triplets-array-whose-sum-equal-zero/

Chapter 60

Find an array element such that all elements are divisible by it

Find an array element such that all elements are divisible by it - GeeksforGeeks

Given an array of numbers, find the number among them such that all numbers are divisible by it. If not possible print -1.

Examples:

```
Input : arr = {25, 20, 5, 10, 100}
Output : 5
Explanation : 5 is an array element
 which divides all numbers.

Input : arr = {9, 3, 6, 2, 15}
Output : -1
Explanation : No numbers are divisible
by any array element.
```

Method 1:(naive)
A normal approach will be to take every element and check for division with all other elements. If all the numbers are divisible then return the number.

C++

```
 // CPP program to find an array element that
// divides all numbers in the array using
// naive approach
#include <bits/stdc++.h>
using namespace std;
```

```
// function to find smallest num
int findSmallest(int a[], int n)
{
    // traverse for all elements
    for (int i = 0; i < n; i++) {

        int j;
        for (j = 0; j < n; j++)
            if (a[j] % a[i])
                break;

        // stores the minimum if
        // it divides all
        if (j == n)
            return a[i];
    }

    return -1;
}

// driver code
int main()
{
    int a[] = { 25, 20, 5, 10, 100 };
    int n = sizeof(a) / sizeof(int);
    cout << findSmallest(a, n);
    return 0;
}
```

Java

```
 // Java program to find an array element
// that divides all numbers in the array
// using naive approach
import java.io.*;

class GFG {

    // function to find smallest num
    static int findSmallest(int a[], int n)
    {
        // traverse for all elements
        for (int i = 0; i < n; i++)
        {

            int j;
            for (j = 0; j < n; j++)
```

```
                if (a[j] % a[i]>=1)
                    break;

            // stores the minimum if
            // it divides all
            if (j == n)
                return a[i];
        }

        return -1;
    }

    // driver code
    public static void main(String args[])
    {
        int a[] = { 25, 20, 5, 10, 100 };
        int n = a.length;
        System.out.println(findSmallest(a, n));
    }
}


// This code is contributed by Nikita Tiwari.
```

Python3

```
 # Python 3 program to find an array
# element that divides all numbers
# in the array using naive approach

# Function to find smallest num
def findSmallest(a, n) :

    # Traverse for all elements
    for i in range(0, n ) :

        for j in range(0, n) :

            if ((a[j] % a[i]) >= 1) :
                break

        # Stores the minimum
        # if it divides all
        if (j == n - 1) :
            return a[i]

    return -1
```

```
# Driver code
a = [ 25, 20, 5, 10, 100 ]
n = len(a)
print(findSmallest(a, n))


# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to find an array element
// that divides all numbers in the array
// using naive approach
using System;

class GFG {

    // function to find smallest num
    static int findSmallest(int []a, int n)
    {
        // traverse for all elements
        for (int i = 0; i < n; i++)
        {

            int j;
            for (j = 0; j < n; j++)
                if (a[j] % a[i] >= 1)
                    break;

            // stores the minimum if
            // it divides all
            if (j == n)
                return a[i];
        }

        return -1;
    }

    // Driver code
    public static void Main()
    {
        int []a = { 25, 20, 5, 10, 100 };
        int n = a.Length;
        Console.WriteLine(findSmallest(a, n));
    }
}
```

```
// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find an array
// element that divides all
// numbers in the array using
// naive approach

// function to find smallest num
function findSmallest($a, $n)
{

    // traverse for all elements
    for ($i = 0; $i < $n; $i++)
    {
        $j;
        for ($j = 0; $j < $n; $j++)
            if ($a[$j] % $a[$i])
                break;

        // stores the minimum if
        // it divides all
        if ($j == $n)
            return $a[$i];
    }

    return -1;
}

    // Driver Code
    $a = array( 25, 20, 5, 10, 100 );
    $n = sizeof($a);
    echo findSmallest($a, $n);

// This code is contributed by nitin mittal
?>
```

Output :

```
5
```

Time Complexity: O(n^2)

Method 2 : (Efficient)
An efficient approach is to find smallest of all numbers, and check if it divides all the other numbers, if yes then the smallest number will be the required number.

C++

```
 // CPP Program to find the smallest number
// that divides all numbers in an array
#include <bits/stdc++.h>
using namespace std;

// function to find smallest num
int findSmallest(int a[], int n)
{
    // Find the smallest element
    int smallest = *min_element(a, a+n);

    // Check if all array elements
    // are divisible by smallest.
    for (int i = 1; i < n; i++)
        if (a[i] % smallest)
            return -1;

    return smallest;
}

// Driver code
int main()
{
    int a[] = { 25, 20, 5, 10, 100 };
    int n = sizeof(a) / sizeof(int);
    cout << findSmallest(a, n);
    return 0;
}
```

Java

```
 // Java Program to find the
// smallest number that divides
// all numbers in an array
import java.io.*;

class GFG {

    // function to find the smallest element
    static int min_element(int a[])
    {
```

```
        int min = Integer.MAX_VALUE, i;
        for (i = 0; i < a.length; i++)
        {
            if (a[i] < min)
                min = a[i];
        }

        return min;
    }

    // function to find smallest num
    static int findSmallest(int a[], int n)
    {
        // Find the smallest element
        int smallest = min_element(a);

        // Check if all array elements
        // are divisible by smallest.
        for (int i = 1; i < n; i++)
        if (a[i] % smallest >= 1)
            return -1;

        return smallest;
    }

    // Driver code
    public static void main(String args[])
    {
        int a[] = {25, 20, 5, 10, 100};
        int n = a.length;
        System.out.println(findSmallest(a, n));
    }
}

// This code is contributed by Nikita Tiwari.
```

Python3

```
 # Python3 Program to find the
# smallest number that divides
# all numbers in an array

# Function to find the smallest element
def min_element(a) :

    m = 10000000

    for i in range(0, len(a)) :
```

```
        if (a[i] < m) :
            m = a[i]

    return m

# Function to find smallest num
def findSmallest(a, n) :

    # Find the smallest element
    smallest = min_element(a)

    # Check if all array elements
    # are divisible by smallest.
    for i in range(1, n) :

        if (a[i] % smallest >= 1) :
            return -1

    return smallest


# Driver code

a = [ 25, 20, 5, 10, 100 ]
n = len(a)
print(findSmallest(a, n))


# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# Program to find the
// smallest number that divides
// all numbers in an array
using System;

class GFG {

    // function to find the smallest element
    static int min_element(int []a)
    {
        int min = int.MaxValue;
        int i;
        for (i = 0; i < a.Length; i++)
        {
            if (a[i] < min)
```

```
                min = a[i];
        }

        return min;
    }

    // function to find smallest num
    static int findSmallest(int []a, int n)
    {
        // Find the smallest element
        int smallest = min_element(a);

        // Check if all array elements
        // are divisible by smallest.
        for (int i = 1; i < n; i++)
        if (a[i] % smallest >= 1)
            return -1;

        return smallest;
    }

    // Driver code
    public static void Main()
    {
        int []a = {25, 20, 5, 10, 100};
        int n = a.Length;
        Console.WriteLine(findSmallest(a, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find the smallest number
// that divides all numbers in an array

// function to find smallest num
function findSmallest($a, $n)
{

    // Find the smallest element
    $smallest = min($a);

    // Check if all array elements
    // are divisible by smallest.
    for ($i = 1; $i < $n; $i++)
```

```
        if ($a[$i] % $smallest)
            return -1;

    return $smallest;
}

    // Driver Code
    $a = array(25, 20, 5, 10, 100);
    $n =count($a);
    echo findSmallest($a, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
5
```

Time Complexity: O(n)

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/number-among-n-numbers-numbers-divisible/

Chapter 61

Find closest number in array

Find closest number in array - GeeksforGeeks

Given an array of sorted integers. We need to find the closest value to the given number. Array may contain duplicate values and negative numbers.

Examples:

```
Input : arr[] = {1, 2, 4, 5, 6, 6, 8, 9}
             Target number = 11
Output : 9
9 is closest to 11 in given array

Input :arr[] = {2, 5, 6, 7, 8, 8, 9};
       Target number = 4
Output : 5
```

A **simple solution** is to traverse through the given array and keep track of absolute difference of current element with every element. Finally return the element that has minimum absolution difference.

An **efficient solution** is to use Binary Search.

C++

```
 // CPP program to find element
// closet to given target.
#include <iostream>
using namespace std;

int getClosest(int, int, int);

// Returns element closest to target in arr[]
```

```
int findClosest(int arr[], int n, int target)
{
    // Corner cases
    if (target <= arr[0])
        return arr[0];
    if (target >= arr[n - 1])
        return arr[n - 1];

    // Doing binary search
    int i = 0, j = n, mid = 0;
    while (i < j) {
        mid = (i + j) / 2;

        if (arr[mid] == target)
            return arr[mid];

        /* If target is less than array element,
            then search in left */
        if (target < arr[mid]) {

            // If target is greater than previous
            // to mid, return closest of two
            if (mid > 0 && target > arr[mid - 1])
                return getClosest(arr[mid - 1],
                                  arr[mid], target);

            /* Repeat for left half */
            j = mid;
        }

        // If target is greater than mid
        else {
            if (mid < n - 1 && target < arr[mid + 1])
                return getClosest(arr[mid],
                                  arr[mid + 1], target);
            // update i
            i = mid + 1;
        }
    }

    // Only single element left after search
    return arr[mid];
}

// Method to compare which one is the more close.
// We find the closest by taking the difference
// between the target and both values. It assumes
// that val2 is greater than val1 and target lies
```

```
// between these two.
int getClosest(int val1, int val2,
               int target)
{
    if (target - val1 >= val2 - target)
        return val2;
    else
        return val1;
}

// Driver code
int main()
{
    int arr[] = { 1, 2, 4, 5, 6, 6, 8, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 11;
    cout << (findClosest(arr, n, target));
}

// This code is contributed bu Smitha Dinesh Semwal
```

Java

```
 // Java program to find element closet to given target.
import java.util.*;
import java.lang.*;
import java.io.*;

class FindClosestNumber {

    // Returns element closest to target in arr[]
    public static int findClosest(int arr[], int target)
    {
        int n = arr.length;

        // Corner cases
        if (target <= arr[0])
            return arr[0];
        if (target >= arr[n - 1])
            return arr[n - 1];

        // Doing binary search
        int i = 0, j = n, mid = 0;
        while (i < j) {
            mid = (i + j) / 2;

            if (arr[mid] == target)
                return arr[mid];
```

```
        /* If target is less than array element,
           then search in left */
        if (target < arr[mid]) {

            // If target is greater than previous
            // to mid, return closest of two
            if (mid > 0 && target > arr[mid - 1])
                return getClosest(arr[mid - 1],
                                  arr[mid], target);

            /* Repeat for left half */
            j = mid;
        }

        // If target is greater than mid
        else {
            if (mid < n-1 && target < arr[mid + 1])
                return getClosest(arr[mid],
                      arr[mid + 1], target);
            i = mid + 1; // update i
        }
    }

    // Only single element left after search
    return arr[mid];
}

// Method to compare which one is the more close
// We find the closest by taking the difference
//  between the target and both values. It assumes
// that val2 is greater than val1 and target lies
// between these two.
public static int getClosest(int val1, int val2,
                                     int target)
{
    if (target - val1 >= val2 - target)
        return val2;
    else
        return val1;
}

// Driver code
public static void main(String[] args)
{
    int arr[] = { 1, 2, 4, 5, 6, 6, 8, 9 };
    int target = 11;
    System.out.println(findClosest(arr, target));
```

```
    }
}
```

Python 3

```
 # Python3 program to find element
# closet to given target.

# Returns element closest to target in arr[]
def findClosest(arr, n, target):

    # Corner cases
    if (target <= arr[0]):
        return arr[0]
    if (target >= arr[n - 1]):
        return arr[n - 1]

    # Doing binary search
    i = 0; j = n; mid = 0
    while (i < j):
        mid = (i + j) / 2

        if (arr[mid] == target):
            return arr[mid]

        # If target is less than array
        # element, then search in left
        if (target < arr[mid]) :

            # If target is greater than previous
            # to mid, return closest of two
            if (mid > 0 and target > arr[mid - 1]):
                return getClosest(arr[mid - 1], arr[mid], target)

            # Repeat for left half
            j = mid

        # If target is greater than mid
        else :
            if (mid < n - 1 and target < arr[mid + 1]):
                return getClosest(arr[mid], arr[mid + 1], target)

            # update i
            i = mid + 1

    # Only single element left after search
    return arr[mid]
```

```
# Method to compare which one is the more close.
# We find the closest by taking the difference
# between the target and both values. It assumes
# that val2 is greater than val1 and target lies
# between these two.
def getClosest(val1, val2, target):

    if (target - val1 >= val2 - target):
        return val2
    else:
        return val1

# Driver code
arr = [1, 2, 4, 5, 6, 6, 8, 9]
n = len(arr)
target = 11
print(findClosest(arr, n, target))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to find element
// closet to given target.
using System;

class GFG
{

    // Returns element closest
    // to target in arr[]
    public static int findClosest(int []arr,
                                  int target)
    {
        int n = arr.Length;

        // Corner cases
        if (target <= arr[0])
            return arr[0];
        if (target >= arr[n - 1])
            return arr[n - 1];

        // Doing binary search
        int i = 0, j = n, mid = 0;
        while (i < j)
        {
            mid = (i + j) / 2;
```

```
        if (arr[mid] == target)
            return arr[mid];

        /* If target is less
        than array element,
        then search in left */
        if (target < arr[mid])
        {

            // If target is greater
            // than previous to mid,
            // return closest of two
            if (mid > 0 && target > arr[mid - 1])
                return getClosest(arr[mid - 1],
                            arr[mid], target);

            /* Repeat for left half */
            j = mid;
        }

        // If target is
        // greater than mid
        else
        {
            if (mid < n-1 && target < arr[mid + 1])
                return getClosest(arr[mid],
                     arr[mid + 1], target);
            i = mid + 1; // update i
        }
    }

    // Only single element
    // left after search
    return arr[mid];
}

// Method to compare which one
// is the more close We find the
// closest by taking the difference
// between the target and both
// values. It assumes that val2 is
// greater than val1 and target
// lies between these two.
public static int getClosest(int val1, int val2,
                             int target)
{
    if (target - val1 >= val2 - target)
```

```
            return val2;
        else
            return val1;
    }

    // Driver code
    public static void Main()
    {
        int []arr = {1, 2, 4, 5,
                     6, 6, 8, 9};
        int target = 11;
        Console.WriteLine(findClosest(arr, target));
    }
}

// This code is contributed by anuj_67.
```

Output:

```
9
```

Improved By : harishkumar88, vt_m

Source

https://www.geeksforgeeks.org/find-closest-number-array/

Chapter 62

Find common elements in three sorted arrays

Find common elements in three sorted arrays - GeeksforGeeks

Given three arrays sorted in non-decreasing order, print all common elements in these arrays.

Examples:

```
ar1[] = {1, 5, 10, 20, 40, 80}
ar2[] = {6, 7, 20, 80, 100}
ar3[] = {3, 4, 15, 20, 30, 70, 80, 120}
Output: 20, 80

ar1[] = {1, 5, 5}
ar2[] = {3, 4, 5, 5, 10}
ar3[] = {5, 5, 10, 20}
Output: 5, 5
```

A simple solution is to first find intersection of two arraysand store the intersection in a temporary array, then find the intersection of third array and temporary array. Time complexity of this solution is O(n1 + n2 + n3) where n1, n2 and n3 are sizes of ar1[], ar2[] and ar3[] respectively.

The above solution requires extra space and two loops, we can find the common elements using a single loop and without extra space. The idea is similar to intersection of two arrays. Like two arrays loop, we run a loop and traverse three arrays.
Let the current element traversed in ar1[] be x, in ar2[] be y and in ar3[] be z. We can have following cases inside the loop.
1) If x, y and z are same, we can simply print any of them as common element and move ahead in all three arrays.

2) Else If x < y, we can move ahead in ar1[] as x cannot be a common element.

3) Else If y and y > z), we can simply move ahead in ar3[] as z cannot be a common element.

Following are implementations of the above idea.

C++

```
 // C++ program to print common elements in three arrays
#include <iostream>
using namespace std;

// This function prints common elements in ar1
void findCommon(int ar1[], int ar2[], int ar3[], int n1, int n2, int n3)
{
    // Initialize starting indexes for ar1[], ar2[] and ar3[]
    int i = 0, j = 0, k = 0;

    // Iterate through three arrays while all arrays have elements
    while (i < n1 && j < n2 && k < n3)
    {
         // If x = y and y = z, print any of them and move ahead
         // in all arrays
         if (ar1[i] == ar2[j] && ar2[j] == ar3[k])
         {   cout << ar1[i] << " ";   i++; j++; k++; }

         // x < y
         else if (ar1[i] < ar2[j])
             i++;

         // y < z
         else if (ar2[j] < ar3[k])
             j++;

         // We reach here when x > y and z < y, i.e., z is smallest
         else
             k++;
    }
}

// Driver program to test above function
int main()
{
    int ar1[] = {1, 5, 10, 20, 40, 80};
    int ar2[] = {6, 7, 20, 80, 100};
    int ar3[] = {3, 4, 15, 20, 30, 70, 80, 120};
    int n1 = sizeof(ar1)/sizeof(ar1[0]);
    int n2 = sizeof(ar2)/sizeof(ar2[0]);
    int n3 = sizeof(ar3)/sizeof(ar3[0]);

    cout << "Common Elements are ";
```

```
    findCommon(ar1, ar2, ar3, n1, n2, n3);
    return 0;
}
```

Java

```
 // Java program to find common elements in three arrays
class FindCommon
{
    // This function prints common elements in ar1
    void findCommon(int ar1[], int ar2[], int ar3[])
    {
        // Initialize starting indexes for ar1[], ar2[] and ar3[]
        int i = 0, j = 0, k = 0;

        // Iterate through three arrays while all arrays have elements
        while (i < ar1.length && j < ar2.length && k < ar3.length)
        {
             // If x = y and y = z, print any of them and move ahead
             // in all arrays
             if (ar1[i] == ar2[j] && ar2[j] == ar3[k])
             {   System.out.print(ar1[i]+" ");   i++; j++; k++; }

             // x < y
             else if (ar1[i] < ar2[j])
                 i++;

             // y < z
             else if (ar2[j] < ar3[k])
                 j++;

             // We reach here when x > y and z < y, i.e., z is smallest
             else
                 k++;
        }
    }

    // Driver code to test above
    public static void main(String args[])
    {
        FindCommon ob = new FindCommon();

        int ar1[] = {1, 5, 10, 20, 40, 80};
        int ar2[] = {6, 7, 20, 80, 100};
        int ar3[] = {3, 4, 15, 20, 30, 70, 80, 120};

        System.out.print("Common elements are ");
        ob.findCommon(ar1, ar2, ar3);
```

```
    }
}

/*This code is contributed by Rajat Mishra */
```

Python

```
 # Python function to print common elements in three sorted arrays
def findCommon(ar1, ar2, ar3, n1, n2, n3):

    # Initialize starting indexes for ar1[], ar2[] and ar3[]
    i, j, k = 0, 0, 0

    # Iterate through three arrays while all arrays have elements
    while (i < n1 and j < n2 and k< n3):

        # If x = y and y = z, print any of them and move ahead
        # in all arrays
        if (ar1[i] == ar2[j] and ar2[j] == ar3[k]):
            print ar1[i],
            i += 1
            j += 1
            k += 1

        # x < y
        elif ar1[i] < ar2[j]:
            i += 1

        # y < z
        elif ar2[j] < ar3[k]:
            j += 1

        # We reach here when x > y and z < y, i.e., z is smallest
        else:
            k += 1

# Driver program to check above function
ar1 = [1, 5, 10, 20, 40, 80]
ar2 = [6, 7, 20, 80, 100]
ar3 = [3, 4, 15, 20, 30, 70, 80, 120]
n1 = len(ar1)
n2 = len(ar2)
n3 = len(ar3)
print "Common elements are",
findCommon(ar1, ar2, ar3, n1, n2, n3)

# This code is contributed by __Devesh Agrawal__
```

C#

```
 // C# program to find common elements in
// three arrays
using System;

class GFG {

    // This function prints common element
    // s in ar1
    static void findCommon(int []ar1, int []ar2,
                                      int []ar3)
    {

        // Initialize starting indexes for
        // ar1[], ar2[] and ar3[]
        int i = 0, j = 0, k = 0;

        // Iterate through three arrays while
        // all arrays have elements
        while (i < ar1.Length && j < ar2.Length
                              && k < ar3.Length)
        {

            // If x = y and y = z, print any of
            // them and move ahead in all arrays
            if (ar1[i] == ar2[j] &&
                               ar2[j] == ar3[k])
            {
                Console.Write(ar1[i] + " ");
                i++;
                j++;
                k++;
            }

            // x < y
            else if (ar1[i] < ar2[j])
                i++;

            // y < z
            else if (ar2[j] < ar3[k])
                j++;

            // We reach here when x > y and
            // z < y, i.e., z is smallest
            else
                k++;
        }
```

```
    }

    // Driver code to test above
    public static void Main()
    {

        int []ar1 = {1, 5, 10, 20, 40, 80};
        int []ar2 = {6, 7, 20, 80, 100};
        int []ar3 = {3, 4, 15, 20, 30,
                            70, 80, 120};

        Console.Write("Common elements are ");

        findCommon(ar1, ar2, ar3);
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// PHP program to print common elements
// in three arrays

// This function prints common elements
// in ar1
function findCommon( $ar1, $ar2, $ar3,
                         $n1, $n2, $n3)
{

    // Initialize starting indexes for
    // ar1[], ar2[] and ar3[]
    $i = 0; $j = 0; $k = 0;

    // Iterate through three arrays while
    // all arrays have elements
    while ($i < $n1 && $j < $n2 && $k < $n3)
    {

        // If x = y and y = z, print any
        // of them and move ahead in all
        // arrays
        if ($ar1[$i] == $ar2[$j] &&
                      $ar2[$j] == $ar3[$k])
        {
            echo $ar1[$i] , " ";
            $i++;
```

```
            $j++;
            $k++;
        }

        // x < y
        else if ($ar1[$i] < $ar2[$j])
            $i++;

        // y < z
        else if ($ar2[$j] < $ar3[$k])
            $j++;

        // We reach here when x > y and
        // z < y, i.e., z is smallest
        else
            $k++;
    }
}

// Driver program to test above function
    $ar1 = array(1, 5, 10, 20, 40, 80);
    $ar2 = array(6, 7, 20, 80, 100);
    $ar3 = array(3, 4, 15, 20, 30, 70,
                                80, 120);
    $n1 = count($ar1);
    $n2 = count($ar2);
    $n3 = count($ar3);

    echo "Common Elements are ";

    findCommon($ar1, $ar2, $ar3,$n1, $n2, $n3);

// This code is contributed by anuj_67.
?>
```

Output:

```
Common Elements are 20 80
```

Time complexity of the above solution is O(n1 + n2 + n3). In worst case, the largest sized array may have all small elements and middle sized array has all middle elements.

This article is compiled by **Rahul Gupta** Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : Sam007, vt_m

Source

https://www.geeksforgeeks.org/find-common-elements-three-sorted-arrays/

Chapter 63

Find duplicate in an array in O(n) and by using O(1) extra space

Find duplicate in an array in O(n) and by using O(1) extra space - GeeksforGeeks

Given an array arr[] containing n + 1 integers where each integer is between 1 and n (inclusive). There is only one duplicate element, find the duplicate element in O(n) time complexity and O(1) space.

Examples :

```
Input  : arr[] = {1, 4, 3, 4, 2}
Output : 4

Input  : arr[] = {1, 3, 2, 1}
Output : 1
```

Approach :
Firstly, the constraints of this problem imply that a cycle must exist. Because each number in an array arr[] is between 1 and n, it will necessarily point to an index that exists. Therefore, the list can be traversed infinitely, which implies that there is a cycle. Additionally, because 0 cannot appear as a value in an array arr[], arr[0] cannot be part of the cycle. Therefore, traversing the array in this manner from arr[0] is equivalent to traversing a cyclic linked list. The problem can be solved just like linked list cycle.

Below is the implementation of above approach:

C++

```
 // CPP code to find the repeated elements
// in the array where every other is present once
#include <iostream>
using namespace std;

// Function to find duplicate
int findDuplicate(int arr[])
{
    // Find the intersection point of
    // the slow and fast.
    int slow = arr[0];
    int fast = arr[0];
    do
    {
        slow = arr[slow];
        fast = arr[arr[fast]];
    } while (slow != fast);

    // Find the "entrance" to the cycle.
    int ptr1 = arr[0];
    int ptr2 = slow;
    while (ptr1 != ptr2)
    {
        ptr1 = arr[ptr1];
        ptr2 = arr[ptr2];
    }

    return ptr1;
}

// Driver code
int main()
{
    int arr[] = { 1, 3, 2, 1 };

    cout << findDuplicate(arr) << endl;

    return 0;
}
```

Java

```
 // Java code to find the repeated
// elements in the array where
// every other is present once
import java.util.*;

class GFG
```

```
{

// Function to find duplicate
public static int findDuplicate(int []arr)
{
    // Find the intersection
    // point of the slow and fast.
    int slow = arr[0];
    int fast = arr[0];
    do
    {
        slow = arr[slow];
        fast = arr[arr[fast]];
    } while (slow != fast);

    // Find the "entrance"
    // to the cycle.
    int ptr1 = arr[0];
    int ptr2 = slow;
    while (ptr1 != ptr2)
    {
        ptr1 = arr[ptr1];
        ptr2 = arr[ptr2];
    }

    return ptr1;
}

// Driver Code
public static void main(String[] args)
{
    int []arr = {1, 3, 2, 1};

    System.out.println("" +
               findDuplicate(arr));

    System.exit(0);
}
}

// This code is contributed
// by Harshit Saini
```

Python3

```
 # Python code to find the
# repeated elements in the
# array where every other
```

```
# is present once

# Function to find duplicate
def findDuplicate(arr):

    # Find the intersection
    # point of the slow and fast.
    slow = arr[0]
    fast = arr[0]
    while True:
        slow = arr[slow]
        fast = arr[arr[fast]]
        if slow == fast:
            break

    # Find the "entrance"
    # to the cycle.
    ptr1 = arr[0]
    ptr2 = slow
    while ptr1 != ptr2:
        ptr1 = arr[ptr1]
        ptr2 = arr[ptr2]

    return ptr1

# Driver code
if __name__ == '__main__':


    arr = [ 1, 3, 2, 1 ]

    print(findDuplicate(arr))


# This code is contributed
# by Harshit Saini
```

C#

```
 // C# code to find the repeated
// elements in the array where
// every other is present once
using System;

class GFG
{

// Function to find duplicate
```

```
public static int findDuplicate(int []arr)
{
    // Find the intersection
    // point of the slow and fast.
    int slow = arr[0];
    int fast = arr[0];
    do
    {
        slow = arr[slow];
        fast = arr[arr[fast]];
    } while (slow != fast);

    // Find the "entrance"
    // to the cycle.
    int ptr1 = arr[0];
    int ptr2 = slow;
    while (ptr1 != ptr2)
    {
        ptr1 = arr[ptr1];
        ptr2 = arr[ptr2];
    }

    return ptr1;
}

// Driver Code
public static void Main()
{
    int[] arr = {1, 3, 2, 1};

    Console.WriteLine("" +
            findDuplicate(arr));

}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

PHP

```
 <?php
// PHP code to find the repeated
// elements in the array where
// every other is present once

// Function to find duplicate
function findDuplicate(&$arr)
```

```
{
    $slow = $arr[0];
    $fast = $arr[0];
    do
    {
        $slow = $arr[$slow];
        $fast = $arr[$arr[$fast]];
    } while ($slow != $fast);

    // Find the "entrance"
    // to the cycle.
    $ptr1 = $arr[0];
    $ptr2 = $slow;
    while ($ptr1 != $ptr2)
    {
        $ptr1 = $arr[$ptr1];
        $ptr2 = $arr[$ptr2];
    }

    return $ptr1;
}

// Driver code
$arr = array(1, 3, 2, 1);
echo " " . findDuplicate($arr);

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output:

```
1
```

Improved By : Harshit Saini, Abby_akku, Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/duplicates-array-using-o1-extra-space-set-3/

Chapter 64

Find element in a sorted array whose frequency is greater than or equal to n/2.

Find element in a sorted array whose frequency is greater than or equal to n/2. - GeeksforGeeks

Given a sorted array of length n, find the number in array that appears more than or equal to n/2 times. It is given that such element always exists.

Examples:

```
Input :  2 3 3 4
Output : 3

Input :  3 4 5 5 5
Output : 5

Input : 1 1 1 2 3
Output : 1
```

To find that number, we traverse the array and check the frequency of every element in array if it is greater than or equals to n/2 but it requires extra space and time complexity will be O(n).
But we can see that the if there is number that comes more than or equal to n/2 times in a sorted array, then that number must be present at the position n/2 i.e. a[n/2].

C++

```
// C++ code to find majority element in a
```

```
// sorted array
#include <iostream>
using namespace std;

int findMajority(int arr[], int n)
{
    return arr[n / 2];
}

int main()
{
    int arr[] = { 1, 2, 2, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findMajority(arr, n);
    return 0;
}
```

Java

```
 // Java code to find majority element in a
// sorted array
public class Test {

    public static int findMajority(int arr[], int n)
    {
        return arr[n / 2];
    }
    public static void main(String args[])
    {
        int arr[] = { 1, 2, 2, 3 };
        int n = arr.length;
        System.out.println(findMajority(arr, n));
    }
}
```

Python 3

```
 # Python 3 code to find
# majority element in a
# sorted array

def findMajority(arr, n):

    return arr[int(n / 2)]

# Driver Code
arr = [1, 2, 2, 3]
```

```
n = len(arr)
print(findMajority(arr, n))

# This code is contributed by Smitha.
```

C#

```
 // C# code to find majority element in a
// sorted array
using System;

public class GFG {

    public static int findMajority(int []arr, int n)
    {
        return arr[n / 2];
    }

    // Driver code
    public static void Main()
    {

        int []arr = { 1, 2, 2, 3 };
        int n = arr.Length;

        Console.WriteLine(findMajority(arr, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP code to find majority
// element in a sorted array

function findMajority($arr, $n)
{
    return $arr[intval($n / 2)];
}

    // Driver Code
    $arr = array(1, 2, 2, 3);
    $n = count($arr);
    echo findMajority($arr, $n);
```

```
// This code is contributed by Sam007
?>
```

Output:

```
2
```

Time Complexity : O(1)

Related Articles :

Majority element in an unsorted array
Check for majority element in a sorted array

Improved By : vt_m, Smitha Dinesh Semwal, Sam007

Source

https://www.geeksforgeeks.org/find-element-sorted-array-whose-frequency-greater-equal-n2/

Chapter 65

Find final value if we double after every successful search in array

Find final value if we double after every successful search in array - GeeksforGeeks

Given an array and an integer k, traverse the array and if the element in array is k, double the value of k and continue traversal. In the end return value of k.

Examples:

```
Input : arr[] = { 2, 3, 4, 10, 8, 1 }, k = 2
Output :16
First k = 2 is found, then we search for 4
which is also found, then we search for 8
which is also found, then we search for 16.

Input : arr[] = { 2, 4, 5, 6, 7 }, k = 3
Output :3
```

1- Traverse each element of an array if arr[i] == k then k = 2 * k
2- Repeat the same process till last element of an array
3- At last Return the value of k

C++

```
 // CPP program to find value if we double
// the value after every successful search
#include <iostream>
using namespace std;
```

```
// Function to Find the value of k
int findValue(int arr[], int n, int k) {

  // Search for k. After every successful
  // search, double k.
  for (int i = 0; i < n; i++)
    if (arr[i] == k)
      k *= 2;

  return k;
}

// Driver's Code
int main() {
  int arr[] = {2, 3, 4, 10, 8, 1}, k = 2;
  int n = sizeof(arr) / sizeof(arr[0]);
  cout << findValue(arr, n, k);
  return 0;
}
```

Java

```
 // Java program to find value
// if we double  the value after
// every successful search

class GFG
{
    // Function to Find the value of k
    static int findValue(int arr[], int n, int k) {

    // Search for k. After every successful
    // search, double k.
    for (int i = 0; i < n; i++)
        if (arr[i] == k)
        k *= 2;

    return k;
    }

    // Driver Code
    public static void main(String[] args)
    {
    int arr[] = {2, 3, 4, 10, 8, 1}, k = 2;
    int n = arr.length;
    System.out.print(findValue(arr, n, k));
    }
```

```
}
// This code is contriubted by
// Smitha Dinesh Semwal
```

Python3

```
 # Python program to find
# value if we double
# the value after every
# successful search

# Function to Find the value of k
def findValue(arr,n,k):

    # Search for k.
    # After every successful
    # search, double k.
    for i in range(n):
        if (arr[i] == k):
            k =k* 2

    return k

# Driver's Code

arr = [2, 3, 4, 10, 8, 1]
k = 2
n = len(arr)

print(findValue(arr, n, k))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to find value
// if we double the value after
// every successful search
using System;

class GFG {

    // Function to Find the value of k
    static int findValue(int []arr, int n, int k) {

        // Search for k. After every successful
```

```
        // search, double k.
        for (int i = 0; i < n; i++)
            if (arr[i] == k)
                k *= 2;

        return k;
    }

    // Driver Code
    public static void Main()
    {
        int []arr = {2, 3, 4, 10, 8, 1};
        int k = 2;
        int n = arr.Length;

        Console.WriteLine(findValue(arr, n, k));
    }
}

// This code is contriubted by vt_m.
```

PHP

```
 <?php
// PHP program to find
// value if we double
// the value after every
// successful search

// Function to Find
// the value of k
function findValue($arr, $n, $k)
{

    // Search for k. After every
    // successful search, double k.
    for ($i = 0; $i < $n; $i++)
        if ($arr[$i] == $k)
        $k *= 2;

    return $k;
}

// Driver Code
$arr = array(2, 3, 4, 10, 8, 1);
$k = 2;
$n = count($arr);
echo findValue($arr, $n, $k);
```

```
// This code is contriubted by anuj_67.
?>
```

```
Output: 16
```

Time Complexity : O(n)

Reference:href=”https://www.geeksforgeeks.org/flipkart-interview-experience-set-35-on-campus-for-sde-1/”

Improved By : vt_m

Source

https://www.geeksforgeeks.org/find-final-value-if-we-double-after-every-successful-search-in-array/

Chapter 66

Find four elements that sum to a given value | Set 1 (n^3 solution)

Find four elements that sum to a given value | Set 1 (n^3 solution) - GeeksforGeeks

Given an array of integers, find all combination of four elements in the array whose sum is equal to a given value X.
For example, if the given array is {10, 2, 3, 4, 5, 9, 7, 8} and X = 23, then your function should print "3 5 7 8" (3 + 5 + 7 + 8 = 23).

A **Naive Solution** is to generate all possible quadruples and compare the sum of every quadruple with X. The following code implements this simple method using four nested loops

C

```
 #include <stdio.h>

/* A naive solution to print all combination of 4 elements in A[]
  with sum equal to X */
void findFourElements(int A[], int n, int X)
{
  // Fix the first element and find other three
  for (int i = 0; i < n-3; i++)
  {
    // Fix the second element and find other two
    for (int j = i+1; j < n-2; j++)
    {
      // Fix the third element and find the fourth
      for (int k = j+1; k < n-1; k++)
      {
```

```
        // find the fourth
        for (int l = k+1; l < n; l++)
           if (A[i] + A[j] + A[k] + A[l] == X)
              printf("%d, %d, %d, %d", A[i], A[j], A[k], A[l]);
      }
    }
  }
}

// Driver program to test above funtion
int main()
{
    int A[] = {10, 20, 30, 40, 1, 2};
    int n = sizeof(A) / sizeof(A[0]);
    int X = 91;
    findFourElements (A, n, X);
    return 0;
}
```

Java

```
 class FindFourElements
{

    /* A naive solution to print all combination of 4 elements in A[]
       with sum equal to X */
    void findFourElements(int A[], int n, int X)
    {
        // Fix the first element and find other three
        for (int i = 0; i < n - 3; i++)
        {
            // Fix the second element and find other two
            for (int j = i + 1; j < n - 2; j++)
            {
                // Fix the third element and find the fourth
                for (int k = j + 1; k < n - 1; k++)
                {
                    // find the fourth
                    for (int l = k + 1; l < n; l++)
                    {
                        if (A[i] + A[j] + A[k] + A[l] == X)
                            System.out.print(A[i]+" "+A[j]+" "+A[k]
                                                            +" "+A[l]);
                    }
                }
            }
        }
    }
```

```
    // Driver program to test above functions
    public static void main(String[] args)
    {
        FindFourElements findfour = new FindFourElements();
        int A[] = {10, 20, 30, 40, 1, 2};
        int n = A.length;
        int X = 91;
        findfour.findFourElements(A, n, X);
    }
}
```

Python3

```
 # A naive solution to print all combination
# of 4 elements in A[] with sum equal to X
def findFourElements(A, n, X):

    # Fix the first element and find
    # other three
    for i in range(0,n-3):

        # Fix the second element and
        # find other two
        for j in range(i+1,n-2):

            # Fix the third element
            # and find the fourth
            for k in range(j+1,n-1):

                # find the fourth
                for l in range(k+1,n):

                    if A[i] + A[j] + A[k] + A[l] == X:
                        print ("%d, %d, %d, %d"
                        %( A[i], A[j], A[k], A[l]))

# Driver program to test above funtion
A = [10, 2, 3, 4, 5, 9, 7, 8]
n = len(A)
X = 23
findFourElements (A, n, X)

# This code is contributed by shreyanshi_arun
```

C#

```
 // C# program for naive solution to
```

```
// print all combination of 4 elements
// in A[] with sum equal to X
using System;

class FindFourElements
{
    void findFourElements(int []A, int n, int X)
    {
        // Fix the first element and find other three
        for (int i = 0; i < n - 3; i++)
        {
            // Fix the second element and find other two
            for (int j = i + 1; j < n - 2; j++)
            {
                // Fix the third element and find the fourth
                for (int k = j + 1; k < n - 1; k++)
                {
                    // find the fourth
                    for (int l = k + 1; l < n; l++)
                    {
                        if (A[i] + A[j] + A[k] + A[l] == X)
                        Console.Write(A[i] + " " + A[j] +
                                " " + A[k] + " " + A[l]);
                    }
                }
            }
        }
    }

    // Driver program to test above functions
    public static void Main()
    {
        FindFourElements findfour = new FindFourElements();
        int []A = {10, 20, 30, 40, 1, 2};
        int n = A.Length;
        int X = 91;
        findfour.findFourElements(A, n, X);
    }
}

// This code is contributed by nitin mittal
```

PHP

```
 <?php
/* A naive solution to print all combination
of 4 elements in A[] with sum equal to X */
```

```
function findFourElements($A, $n, $X)
{
    // Fix the first element and find other
    // three
    for ($i = 0; $i < $n-3; $i++)
    {
        // Fix the second element and find
        // other two
        for ($j = $i+1; $j < $n-2; $j++)
        {
            // Fix the third element and
            // find the fourth
            for ($k = $j+1; $k < $n-1; $k++)
            {
                // find the fourth
                for ($l = $k+1; $l < $n; $l++)
                    if ($A[$i] + $A[$j] +
                        $A[$k] + $A[$l] == $X)

                        echo $A[$i], ", " , $A[$j],
                        ", ", $A[$k], ", ", $A[$l];
            }
        }
    }
}

// Driver program to test above funtion
    $A = array (10, 20, 30, 40, 1, 2);
    $n = sizeof($A) ;
    $X = 91;
    findFourElements ($A, $n, $X);

// This code is contributed by m_kit
?>
```

Output:

```
20, 30, 40, 1
```

Time Complexity: O(n^4)

The time complexity can be improved to O(n^3) with the ***use of sorting*** *as a preprocessing step, and then using method 1 of this post to reduce a loop.*

Following are the detailed steps.
1) Sort the input array.
2) Fix the first element as A[i] where i is from 0 to n–3. After fixing the first element of quadruple, fix the second element as A[j] where j varies from i+1 to n-2. Find remaining two elements in O(n) time, using the method 1 of this post

Following is the implementation of O(n^3) solution.

C

```
 # include <stdio.h>
# include <stdlib.h>

/* Following function is needed for library function qsort(). Refer
   http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
int compare (const void *a, const void * b)
{  return ( *(int *)a - *(int *)b ); }

/* A sorting based solution to print all combination of 4 elements in A[]
   with sum equal to X */
void find4Numbers(int A[], int n, int X)
{
    int l, r;

    // Sort the array in increasing order, using library
    // function for quick sort
    qsort (A, n, sizeof(A[0]), compare);

    /* Now fix the first 2 elements one by one and find
       the other two elements */
    for (int i = 0; i < n - 3; i++)
    {
        for (int j = i+1; j < n - 2; j++)
        {
            // Initialize two variables as indexes of the first and last
            // elements in the remaining elements
            l = j + 1;
            r = n-1;

            // To find the remaining two elements, move the index
            // variables (l & r) toward each other.
            while (l < r)
            {
                if( A[i] + A[j] + A[l] + A[r] == X)
                {
                   printf("%d, %d, %d, %d", A[i], A[j],
                                            A[l], A[r]);
                   l++; r--;
                }
                else if (A[i] + A[j] + A[l] + A[r] < X)
                    l++;
                else // A[i] + A[j] + A[l] + A[r] > X
                    r--;
            } // end of while
```

```
        } // end of inner for loop
    } // end of outer for loop
}

/* Driver program to test above function */
int main()
{
    int A[] = {1, 4, 45, 6, 10, 12};
    int X = 21;
    int n = sizeof(A)/sizeof(A[0]);
    find4Numbers(A, n, X);
    return 0;
}
```

Java

```
 import java.util.Arrays;


class FindFourElements
{
    /* A sorting based solution to print all combination of 4 elements in A[]
       with sum equal to X */
    void find4Numbers(int A[], int n, int X)
    {
        int l, r;

        // Sort the array in increasing order, using library
        // function for quick sort
        Arrays.sort(A);

        /* Now fix the first 2 elements one by one and find
           the other two elements */
        for (int i = 0; i < n - 3; i++)
        {
            for (int j = i + 1; j < n - 2; j++)
            {
                // Initialize two variables as indexes of the first and last
                // elements in the remaining elements
                l = j + 1;
                r = n - 1;

                // To find the remaining two elements, move the index
                // variables (l & r) toward each other.
                while (l < r)
                {
                    if (A[i] + A[j] + A[l] + A[r] == X)
                    {
```

```
                        System.out.println(A[i]+" "+A[j]+" "+A[l]+" "+A[r]);
                        l++;
                        r--;
                    }
                    else if (A[i] + A[j] + A[l] + A[r] < X)
                        l++;
                    else // A[i] + A[j] + A[l] + A[r] > X
                        r--;
                } // end of while
            } // end of inner for loop
        } // end of outer for loop
    }

    // Driver program to test above functions
    public static void main(String[] args)
    {
        FindFourElements findfour = new FindFourElements();
        int A[] = {1, 4, 45, 6, 10, 12};
        int n = A.length;
        int X = 21;
        findfour.find4Numbers(A, n, X);
    }
}

// This code has been contributed by Mayank Jaiswal
```

C#

```
 // C# program for to  print all combination
// of 4 elements in A[] with sum equal to X
using System;

class FindFourElements
{
    // A sorting based solution to print all
    // combination of 4 elements in A[] with
    // sum equal to X
    void find4Numbers(int []A, int n, int X)
    {
        int l, r;

        // Sort the array in increasing order,
        // using library function for quick sort
        Array.Sort(A);

        /* Now fix the first 2 elements one by one
           and find the other two elements */
        for (int i = 0; i < n - 3; i++)
```

```
        {
            for (int j = i + 1; j < n - 2; j++)
            {
                // Initialize two variables as indexes of
                // the first and last elements in the
                // remaining elements
                l = j + 1;
                r = n - 1;

                // To find the remaining two elements, move the
                // index variables (l & r) toward each other.
                while (l < r)
                {
                    if (A[i] + A[j] + A[l] + A[r] == X)
                    {
                        Console.Write(A[i] + " " + A[j] +
                                " " + A[l] + " " + A[r]);
                        l++;
                        r--;
                    }
                    else if (A[i] + A[j] + A[l] + A[r] < X)
                        l++;

                    else // A[i] + A[j] + A[l] + A[r] > X
                        r--;

                } // end of while

            } // end of inner for loop

        } // end of outer for loop
    }

    // Driver program to test above functions
    public static void Main()
    {
        FindFourElements findfour = new FindFourElements();
        int []A = {1, 4, 45, 6, 10, 12};
        int n = A.Length;
        int X = 21;
        findfour.find4Numbers(A, n, X);
    }
}

// This code has been contributed by nitin mittal
```

PHP

```
 <?php
// PHP program for to print all
// combination of 4 elements in
// A[] with sum equal to X

// A sorting based solution to
// print all combination of 4
// elements in A[] with sum
// equal to X
function find4Numbers($A, $n, $X)
{
    $l; $r;

    // Sort the array in increasing
    // order, using library function
    // for quick sort
    sort($A);

    // Now fix the first 2 elements
    // one by one and find the other
    // two elements
    for ($i = 0; $i < $n - 3; $i++)
    {
        for ($j = $i + 1; $j < $n - 2; $j++)
        {
            // Initialize two variables
            // as indexes of the first
            // and last elements in the
            // remaining elements
            $l = $j + 1;
            $r = $n - 1;

            // To find the remaining two
            // elements, move the index
            // variables (l & r) towards
            // each other.
            while ($l < $r)
            {
                if ($A[$i] + $A[$j] +
                    $A[$l] + $A[$r] == $X)
                {
                    echo $A[$i] , " " , $A[$j] ,
                                  " " , $A[$l] ,
                                  " " , $A[$r];
                    $l++;
                    $r--;
                }
                else if ($A[$i] + $A[$j] +
```

```
                        $A[$l] + $A[$r] < $X)
                    $l++;

                // A[i] + A[j] + A[l] + A[r] > X
                else
                    $r--;

            }
        }
    }
}

// Driver Code
$A = array(1, 4, 45, 6, 10, 12);
$n = count($A);
$X = 21;
find4Numbers($A, $n, $X);

// This code is contributed
// by nitin mittal
?>
```

Output :

```
1, 4, 6, 10
```

Time Complexity : O(n^3)

This problem can also be solved in O(n^2Logn) complexity. Please refer below post for details

Find four elements that sum to a given value | Set 2 (O(n^2Logn) Solution)

Improved By : nitin mittal, jit_t, vt_m

Source

https://www.geeksforgeeks.org/find-four-numbers-with-sum-equal-to-given-sum/

Chapter 67

Find four missing numbers in an array containing elements from 1 to N

Find four missing numbers in an array containing elements from 1 to N - GeeksforGeeks

Given an array of unique integers where each integer of the given array lies in the range [1, N]. The size of array is (N-4). No Single element is repeated. Hence four numbers from 1 to N are missing in the array. Find the 4 missing numbers in sorted order.

Examples:

```
Input : arr[] = {2, 5, 6, 3, 9}
Output : 1 4 7 8

Input : arr[] = {1, 7, 3, 13, 5, 10, 8, 4, 9}
Output : 2 6 11 12
```

A simple O(N) solution is to use an auxiliary array of size N to mark visited elements. Traverse the input array and mark elements in the auxiliary array. Finally print all those indexes that are not marked.

How to solve with O(1) auxiliary space?
We initialize an array named **helper** of length 4 in order to compensate the 4 missing numbers and fill them with zero. Then we iterate from i=0 to i < length_of_array of the given array and take the Absolute of the of the i-th element and store it in a variable named **temp**. Now we will check:

1. If the element's absolute value is less than the length of the input array then we will multiply the array[temp] element with -1 (in order to mark the visited element).

2. If the element's absolute value is greater than the length of the input array then we will put the value of helper[temp%array.length] element with -1 (in order to mark the visited element).

Then we iterate over input array and those index whose value is still greater than zero then those elements were not encountered in the input array. So we print the **(index+1)** value of the index whose element is greater than zero.
Then we will iterate over helper array and those index whose value is still greater than zero then those elements were not encountered in the input array. So we print the **(index+array.length+1)** value of the index whose element is greater than zero.

C++

```
 // CPP program to find missing 4 elements
// in an array of size N where elements are
// in range from 1 to N+4.
#include <bits/stdc++.h>
using namespace std;

// Finds missing 4 numbers in O(N) time
// and O(1) auxiliary space.
void missing4(int arr[], int n)
{
    // To keep track of 4 possible numbers
    // greater than length of input array
    // In Java, helper is automatically
    // initialized as 0.
    int helper[4];

    // Traverse the input array and mark
    // visited elements either by marking
    // them as negative in arr[] or in
    // helper[].
    for (int i = 0; i < n; i++) {
        int temp = abs(arr[i]);

        // If element is smaller than or
        // equal to length, mark its
        // presence in arr[]
        if (temp <= n)
            arr[temp - 1] *= (-1);

        // Mark presence in helper[]
        else if (temp > n) {
            if (temp % n != 0)
                helper[temp % n - 1] = -1;
            else
                helper[(temp % n) + n - 1] = -1;
```

```
        }
    }

    // Print all those elements whose presence
    // is not marked.
    for (int i = 0; i < n; i++)
        if (arr[i] > 0)
            cout << (i + 1) << " ";
    for (int i = 0; i < 4; i++)
        if (helper[i] >= 0)
            cout << (n + i + 1) << " ";

    return;
}

// Driver code
int main()
{
    int arr[] = { 1, 7, 3, 12, 5, 10, 8, 4, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);
    missing4(arr, n);
    return 0;
}

// This code is contributed by Nikita Tiwari.
```

Java

```
 // Java program to find missing 4 elements
// in an array of size N where elements are
// in range from 1 to N+4.
class Missing4 {

    // Finds missing 4 numbers in O(N) time
    // and O(1) auxiliary space.
    public static void missing4(int[] arr)
    {
        // To keep track of 4 possible numbers
        // greater than length of input array
        // In Java, helper is automatically
        // initialized as 0.
        int[] helper = new int[4];

        // Traverse the input array and mark
        // visited elements either by marking
        // them as negative in arr[] or in
        // helper[].
        for (int i = 0; i < arr.length; i++) {
```

```
            int temp = Math.abs(arr[i]);

            // If element is smaller than or
            // equal to length, mark its
            // presence in arr[]
            if (temp <= arr.length)
                arr[temp - 1] *= (-1);

             // Mark presence in helper[]
             else if (temp > arr.length) {
                if (temp % arr.length != 0)
                    helper[temp % arr.length - 1] = -1;
                else
                    helper[(temp % arr.length) +
                                   arr.length - 1] = -1;
            }
        }

        // Print all those elements whose presence
        // is not marked.
        for (int i = 0; i < arr.length; i++)
            if (arr[i] > 0)
                System.out.print(i + 1 + " ");
        for (int i = 0; i < helper.length; i++)
            if (helper[i] >= 0)
                System.out.print(arr.length + i + 1 + " ");

        return;
    }

    // Driver code
    public static void main(String[] args)
    {
        int[] arr = { 1, 7, 3, 12, 5, 10, 8, 4, 9 };
        missing4(arr);
    }
}
```

Python3

```
 # Python 3 program to find missing 4 elements
# in an array of size N where elements are
# in range from 1 to N+4.

# Finds missing 4 numbers in O(N) time
# and O(1) auxiliary space.
def missing4( arr) :
```

```
    # To keep track of 4 possible numbers
    # greater than length of input array
    # In Java, helper is automatically
    # initialized as 0.
    helper = [0]*4

    # Traverse the input array and mark
    # visited elements either by marking
    # them as negative in arr[] or in
    # helper[].
    for i in range(0,len(arr)) :
        temp = abs(arr[i])

        # If element is smaller than or
        # equal to length, mark its
        # presence in arr[]
        if (temp <= len(arr)) :
            arr[temp - 1] = arr[temp - 1] * (-1)

        # Mark presence in helper[]
        elif (temp > len(arr)) :
            if (temp % len(arr)) :
                helper[temp % len(arr) - 1] = -1
            else :
                helper[(temp % len(arr)) +len(arr) - 1] = -1


    # Print all those elements whose presence
    # is not marked.
    for i in range(0, len(arr) ) :
        if (arr[i] > 0) :
            print((i + 1) , end=" ")
    for i in range(0, len(helper)) :
        if (helper[i] >= 0) :
            print((len(arr) + i + 1) , end=" ")


# Driver code
arr = [ 1, 7, 3, 12, 5, 10, 8, 4, 9 ]
missing4(arr)


# This code is conributed
# by Nikita Tiwari.
```

C#

```
 // C# program to find missing 4 elements
```

```
// in an array of size N where elements are
// in range from 1 to N+4.
using System;

class Missing4 {

    // Finds missing 4 numbers in O(N) time
    // and O(1) auxiliary space.
    public static void missing4(int[] arr)
    {
        // To keep track of 4 possible numbers
        // greater than length of input array
        // In Java, helper is automatically
        // initialized as 0.
        int[] helper = new int[4];

        // Traverse the input array and mark
        // visited elements either by marking
        // them as negative in arr[] or in
        // helper[].
        for (int i = 0; i < arr.Length; i++) {
            int temp = Math.Abs(arr[i]);

            // If element is smaller than or
            // equal to length, mark its
            // presence in arr[]
            if (temp <= arr.Length)
                arr[temp - 1] *= (-1);

             // Mark presence in helper[]
             else if (temp > arr.Length) {
                if (temp % arr.Length != 0)
                    helper[temp % arr.Length - 1] = -1;
                else
                    helper[(temp % arr.Length) +
                                   arr.Length - 1] = -1;
            }
        }

        // Print all those elements whose presence
        // is not marked.
        for (int i = 0; i < arr.Length; i++)
            if (arr[i] > 0)
                Console.Write(i + 1 + " ");
        for (int i = 0; i < helper.Length; i++)
            if (helper[i] >= 0)
                Console.Write(arr.Length + i + 1 + " ");
```

```
        return;
    }

    // Driver code
    public static void Main()
    {
        int[] arr = { 1, 7, 3, 12, 5, 10, 8, 4, 9 };
        missing4(arr);
    }
}

//This code is contributed by Anant Agarwal.
```

Output:

```
2 6 11 13
```

Time Complexity: O(N)
Auxiliary Space : O(1)

Source

https://www.geeksforgeeks.org/find-four-missing-numbers-array-containing-elements-1-n/

Chapter 68

Find if given number is sum of first n natural numbers

Find if given number is sum of first n natural numbers - GeeksforGeeks

Given a number s (1 <= s <= 1000000000). If this number is the sum of first n natural number then print, otherwise print -1.

Examples :

```
Input : s = 10
Output : n = 4
1 + 2 + 3 + 4 = 10

Input : s = 17
Output : n -1
17 can't be expressed as a
sum of consecutive from 1.
```

Chapter 69

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

Method 1 (Simple):

Start adding numbers from i = 1 to n.
a) Check if sum is equal to n, return i.
b) Else if sum > n, return -1.

C++

```
 // C++ program for above implementation
#include <iostream>
using namespace std;

// Function to find no. of elements
// to be added from 1 to get n
int findS(int s)
{
    int sum = 0;

    // Start adding numbers from 1
    for (int n = 1; sum < s; n++) {
        sum += n;

        // If sum becomes equal to s
        // return n
        if (sum == s)
            return n;
    }
```

```
    return -1;
}

// Drivers code
int main()
{
    int s = 15;
    int n = findS(s);
    n == -1 ? cout << "-1"
            : cout << n;

    return 0;
}
```

Java

```
 // Java program for above implementation
class GFG {

    // Function to find no. of elements
    // to be added from 1 to get n
    static int findS(int s)
    {
        int sum = 0;

        // Start adding numbers from 1
        for (int n = 1; sum < s; n++)
        {
            sum += n;

            // If sum becomes equal to s
            // return n
            if (sum == s)
                return n;
        }

        return -1;
    }

    // Drivers code
    public static void main(String[]args)
    {

        int s = 15;
        int n = findS(s);
        if(n == -1)
            System.out.println("-1");
```

```
        else
            System.out.println(n);
    }
}

//This code is contributed by Azkia Anam.
```

Python3

```
 # Python3 program to check if
# given number is sum of first n
# natural numbers

# Function to find no. of elements
# to be added from 1 to get n
def findS (s):
    _sum = 0
    n = 1

    # Start adding numbers from 1
    while(_sum < s):
        _sum += n
        n+=1
    n-=1

    # If sum becomes equal to s
    # return n
    if _sum == s:
        return n
    return -1

# Driver code
s = 15
n = findS (s)
if n == -1:
    print("-1")
else:
    print(n)

# This code is contributed by "Abhishek Sharma 44".
```

C#

```
 // C# program for above implementation
using System;

class GFG {
```

```
    // Function to find no. of elements
    // to be added from 1 to get n
    static int findS(int s)
    {
        int sum = 0;

        // Start adding numbers from 1
        for (int n = 1; sum < s; n++)
        {
            sum += n;

            // If sum becomes equal to s
            // return n
            if (sum == s)
                return n;
        }

        return -1;
    }

    // Drivers code
    public static void Main()
    {

        int s = 15;
        int n = findS(s);

        if(n == -1)
            Console.WriteLine("-1");
        else
            Console.WriteLine(n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program for above implementation

// Function to find no. of elements
// to be added from 1 to get n
function findS($s)
{
    $sum = 0;
```

```
    // Start adding numbers from 1
    for ($n = 1; $sum < $s; $n++)
    {
        $sum += $n;

        // If sum becomes equal
        // to s return n
        if ($sum == $s)
            return $n;
    }

    return -1;
}

// Drivers code
$s = 15;
$n = findS($s);
if($n == -1)
echo "-1";
else
echo $n;

// This code is contributed by Sam007
?>
```

Output :

```
5
```

Time Complexity : O(s)
where s are the no. of consecutive numbers from 1 to s

Method 2 (Binary Search):

```
1- Initialize l = 1 and r = n / 2.
2- Apply binary search from l to r.
  a) Find mid = (l+r) / 2
  b) Find sum from 1 to mid using formula
         mid*(mid+1)/2
  c) If sum of mid natural numbers is equal
      to n, return mid.
  d) Else if sum > n, r = mid - 1.
  e) Else sum < n, l = mid + 1.
3- Return -1, if not possible.
```

C++

```
 // C++ program for finding s such
// that sum from 1 to s equals to n
#include <iostream>
using namespace std;

// Function to find no. of elements
// to be added to get s
int findS(int s)
{
    int l = 1, r = s / 2;

    // Apply Binary search
    while (l <= r) {

        // Find mid
        int mid = (l + r) / 2;

        // find sum of 1 to mid natural numbers
        // using formula
        int sum = mid * (mid + 1) / 2;

        // If sum is equal to n
        // return mid
        if (sum == s)
            return mid;

        // If greater than n
        // do r = mid-1
        else if (sum > s)
            r = mid - 1;

        // else do l = mid + 1
        else
            l = mid + 1;
    }

    // If not possible, return -1
    return -1;
}

// Drivers code
int main()
{
    int s = 15;
    int n = findS(s);
    n == -1 ? cout << "-1"
            : cout << n;
```

```
    return 0;
}
```

Java

```
 // java program for finding s such
// that sum from 1 to s equals to n
import java.io.*;

public class GFG {

    // Function to find no. of elements
    // to be added to get s
    static int findS(int s)
    {
        int l = 1, r = s / 2;

        // Apply Binary search
        while (l <= r) {

            // Find mid
            int mid = (l + r) / 2;

            // find sum of 1 to mid natural
            // numbers using formula
            int sum = mid * (mid + 1) / 2;

            // If sum is equal to n
            // return mid
            if (sum == s)
                return mid;

            // If greater than n
            // do r = mid-1
            else if (sum > s)
                r = mid - 1;

            // else do l = mid + 1
            else
                l = mid + 1;
        }

        // If not possible, return -1
        return -1;
    }

    // Drivers code
    static public void main (String[] args)
```

```
    {
        int s = 15;
        int n = findS(s);

        if(n==-1)
            System.out.println("-1");
        else
            System.out.println(n);
    }
}

// This code is contributed by vt_m.
```

Python3

```
 # python program for finding s such
# that sum from 1 to s equals to n

# Function to find no. of elements
# to be added to get s
def findS(s):

    l = 1
    r = int(s / 2)

    # Apply Binary search
    while (l <= r) :
        # Find mid
        mid = int((l + r) / 2)

        # find sum of 1 to mid natural numbers
        # using formula
        sum = int(mid * (mid + 1) / 2)

        # If sum is equal to n
        # return mid
        if (sum == s):
            return mid

        # If greater than n
        # do r = mid-1
        elif (sum > s):
            r = mid - 1

        # else do l = mid + 1
        else:
            l = mid + 1
```

```
    # If not possible, return -1
    return -1


s = 15
n = findS(s)
if(n == -1):
    print( "-1")
else:
    print( n )

# This code is contributed by Sam007
```

C#

```
 // C# program for finding s such
// that sum from 1 to s equals to n
using System;

public class GFG {

    // Function to find no. of elements
    // to be added to get s
    static int findS(int s)
    {
        int l = 1, r = s / 2;

        // Apply Binary search
        while (l <= r) {

            // Find mid
            int mid = (l + r) / 2;

            // find sum of 1 to mid natural
            // numbers using formula
            int sum = mid * (mid + 1) / 2;

            // If sum is equal to n
            // return mid
            if (sum == s)
                return mid;

            // If greater than n
            // do r = mid-1
            else if (sum > s)
                r = mid - 1;
```

```
            // else do l = mid + 1
            else
                l = mid + 1;
        }

        // If not possible, return -1
        return -1;
    }

    // Drivers code
    static public void Main ()
    {
        int s = 15;
        int n = findS(s);

        if(n==-1)
            Console.WriteLine("-1");
        else
            Console.WriteLine(n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program for finding
// s such that sum from 1
// to s equals to n

// Function to find no.
// of elements to be added
// to get s
function findS($s)
{
    $l = 1;
    $r = (int)$s / 2;

    // Apply Binary search
    while ($l <= $r)
    {

        // Find mid
        $mid = (int)(($l + $r) / 2);

        // find sum of 1 to mid natural
        // numbers using formula
```

```
        $sum = (int)($mid *
                    ($mid + 1) / 2);

        // If sum is equal to n
        // return mid
        if ($sum == $s)
            return $mid;

        // If greater than n
        // do r = mid-1
        else if ($sum > $s)
            $r = $mid - 1;

        // else do l = mid + 1
        else
            $l = $mid + 1;
    }

    // If not possible,
    // return -1
    return -1;
}

// Drivers code
$s = 15;
$n = findS($s);
if($n == -1 )
echo "-1";
else
echo $n;

// This code is contributed by Sam007
?>
```

Output :

```
5
```

Time Complexity : O(log n)

Improved By : vt_m, Sam007

Source

https://www.geeksforgeeks.org/find-given-number-sum-first-n-natural-numbers/

Chapter 70

Find index of an extra element present in one sorted array

Find index of an extra element present in one sorted array - GeeksforGeeks

Given two sorted arrays. There is only 1 difference between the arrays. First array has one element extra added in between. Find the index of the extra element.

Examples :

```
Input : {2, 4, 6, 8, 9, 10, 12};
        {2, 4, 6, 8, 10, 12};
Output : 4
The first array has an extra element 9.
The extra element is present at index 4.

Input :  {3, 5, 7, 9, 11, 13}
         {3, 5, 7, 11, 13}
Output :  3
```

Method 1 (Basic)

The basic method is to iterate through the whole second array and check element by element if they are different.

C++

```
 // C++ program to find an extra
// element present in arr1[]
#include <iostream>
using namespace std;
```

```
// Returns index of extra element
// in arr1[]. n is size of arr2[].
// Size of arr1[] is n-1.
int findExtra(int arr1[],
              int arr2[], int n)
{
for (int i = 0; i < n; i++)
    if (arr1[i] != arr2[i])
        return i;

return n;
}

// Driver code
int main()
{
    int arr1[] = {2, 4, 6, 8,
                  10, 12, 13};
    int arr2[] = {2, 4, 6,
                  8, 10, 12};
    int n = sizeof(arr2) / sizeof(arr2[0]);

    // Solve is passed both arrays
    cout << findExtra(arr1, arr2, n);
    return 0;
}
```

Java

```
 // Java program to find an extra
// element present in arr1[]
class GFG
{

    // Returns index of extra element
    // in arr1[]. n is size of arr2[].
    // Size of arr1[] is n-1.
    static int findExtra(int arr1[],
                         int arr2[], int n)
    {
    for (int i = 0; i < n; i++)
        if (arr1[i] != arr2[i])
            return i;

    return n;
    }

    // Driver Code
```

```
    public static void main (String[] args)
    {
        int arr1[] = {2, 4, 6, 8,
                      10, 12, 13};
        int arr2[] = {2, 4, 6,
                      8, 10, 12};
        int n = arr2.length;

        // Solve is passed both arrays
        System.out.println(findExtra(arr1,
                                     arr2, n));
    }
}

// This code is contributed by Harsh Agarwal
```

Python3

```
 # Python 3 program to find an
# extra element present in arr1[]


# Returns index of extra .
# element in arr1[] n is
# size of arr2[]. Size of
# arr1[] is n-1.
def findExtra(arr1, arr2, n) :
    for i in range(0, n) :
        if (arr1[i] != arr2[i]) :
            return i

    return n


# Driver code
arr1 = [2, 4, 6, 8,  10, 12, 13]
arr2 = [2, 4, 6, 8, 10, 12]
n = len(arr2)

# Solve is passed both arrays
print(findExtra(arr1, arr2, n))

# This code is contributed
# by Nikita Tiwari.
```

C#

```
 // C# program to find an extra
```

```
// element present in arr1[]
using System;

class GfG
{

    // Returns index of extra
    // element in arr1[]. n is
    // size of arr2[]. Size of
    // arr1[] is n-1.
    static int findExtra(int []arr1,
                         int []arr2, int n)
    {
        for (int i = 0; i < n; i++)
            if (arr1[i] != arr2[i])
                return i;

        return n;
    }

    // Driver code
    public static void Main ()
    {
        int []arr1 = {2, 4, 6, 8,
                      10, 12, 13};
        int []arr2 = {2, 4, 6,
                      8, 10, 12};
        int n = arr2.Length;

        // Solve is passed both arrays
        Console.Write(findExtra(arr1, arr2, n));
    }
}

// This code is contributed by parashar.
```

PHP

```
 <?php
// PHP program to find an extra
// element present in arr1[]

// Returns index of extra element
// in arr1[]. n is size of arr2[].
// Size of arr1[] is n-1.
function findExtra($arr1,
                   $arr2, $n)
{
```

```
for ($i = 0; $i < $n; $i++)
    if ($arr1[$i] != $arr2[$i])
        return $i;

return $n;
}

// Driver code
$arr1 = array (2, 4, 6, 8,
               10, 12, 13);
$arr2 = array(2, 4, 6,
              8, 10, 12);
$n = sizeof($arr2);

// Solve is passed
// both arrays
echo findExtra($arr1, $arr2, $n);

// This code is contributed by ajit
?>
```

Output :

```
 6
```

Time complexity : O(n)

Method 2 (Using Binary search)

We use binary search to check whether the same indices elements are different & reduce our search by a factor of 2 in each step.

C++

```
 // C++ program to find an extra
// element present in arr1[]
#include <iostream>
using namespace std;

// Returns index of extra element
// in arr1[]. n is size of arr2[].
// Size of arr1[] is n-1.
int findExtra(int arr1[],
              int arr2[], int n)
{
```

```
    // Initialize result
    int index = n;

    // left and right are end
    // points denoting the current range.
    int left = 0, right = n - 1;
    while (left <= right)
    {
        int mid = (left + right) / 2;

        // If middle element is same
        // of both arrays, it means
        // that extra element is after
        // mid so we update left to mid+1
        if (arr2[mid] == arr1[mid])
            left = mid + 1;

        // If middle element is different
        // of the arrays, it means that
        // the index we are searching for
        // is either mid, or before mid.
        // Hence we update right to mid-1.
        else
        {
            index = mid;
            right = mid - 1;
        }
    }

    // when right is greater than
    // left our search is complete.
    return index;
}

// Driver code
int main()
{
    int arr1[] = {2, 4, 6, 8, 10, 12, 13};
    int arr2[] = {2, 4, 6, 8, 10, 12};
    int n = sizeof(arr2) / sizeof(arr2[0]);

    // Solve is passed both arrays
    cout << findExtra(arr1, arr2, n);
    return 0;
}
```

Java

```
 // Java program to find an extra
// element present in arr1[]
class GFG
{
    // Returns index of extra element
    // in arr1[]. n is size of arr2[].
    // Size of arr1[] is n-1.
    static int findExtra(int arr1[],
                         int arr2[], int n)
    {
        // Initialize result
        int index = n;

        // left and right are end
        // points denoting the current range.
        int left = 0, right = n - 1;
        while (left <= right)
        {
            int mid = (left+right) / 2;

            // If middle element is same
            // of both arrays, it means
            // that extra element is after
            // mid so we update left to mid+1
            if (arr2[mid] == arr1[mid])
                left = mid + 1;

            // If middle element is different
            // of the arrays, it means that
            // the index we are searching for
            // is either mid, or before mid.
            // Hence we update right to mid-1.
            else
            {
                index = mid;
                right = mid - 1;
            }
        }

        // when right is greater than
        // left, our search is complete.
        return index;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int arr1[] = {2, 4, 6, 8, 10, 12,13};
```

```
        int arr2[] = {2, 4, 6, 8, 10, 12};
        int n = arr2.length;

        // Solve is passed both arrays
        System.out.println(findExtra(arr1, arr2, n));
    }
}

// This code is contributed by Harsh Agarwal
```

Python3

```
 # Python3 program to find an extra
# element present in arr1[]

# Returns index of extra element
# in arr1[]. n is size of arr2[].
# Size of arr1[] is n-1.
def findExtra(arr1, arr2, n) :

    index = n # Initialize result

    # left and right are end points
    # denoting the current range.
    left = 0
    right = n - 1
    while (left <= right) :
        mid = (int)((left + right) / 2)

        # If middle element is same
        # of both arrays, it means
        # that extra element is after
        # mid so we update left to
        # mid + 1
        if (arr2[mid] == arr1[mid]) :
            left = mid + 1

        # If middle element is different
        # of the arrays, it means that
        # the index we are searching for
        # is either mid, or before mid.
        # Hence we update right to mid-1.
        else :
            index = mid
            right = mid - 1

    # when right is greater than left our
    # search is complete.
```

```
    return index

# Driver code
arr1 = [2, 4, 6, 8, 10, 12, 13]
arr2 = [2, 4, 6, 8, 10, 12]
n = len(arr2)

# Solve is passed both arrays
print(findExtra(arr1, arr2, n))

# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to find an extra
// element present in arr1[]
using System;

class GFG {

    // Returns index of extra
    // element in arr1[]. n is
    // size of arr2[].
    // Size of arr1[] is
    // n - 1.
    static int findExtra(int []arr1,
                         int []arr2,
                         int n)
    {

        // Initialize result
        int index = n;

        // left and right are
        // end points denoting
        // the current range.
        int left = 0, right = n - 1;
        while (left <= right)
        {
            int mid = (left+right) / 2;

            // If middle element is
            // same of both arrays,
            // it means that extra
            // element is after mid
            // so we update left
            // to mid + 1
            if (arr2[mid] == arr1[mid])
```

```
                left = mid + 1;

            // If middle element is
            // different of the arrays,
            // it means that the index
            // we are searching for is
            // either mid, or before mid.
            // Hence we update right to mid-1.
            else
            {
                index = mid;
                right = mid - 1;
            }
        }

        // when right is greater
        // than left our
        // search is complete.
        return index;
    }

    // Driver Code
    public static void Main ()
    {
        int []arr1 = {2, 4, 6, 8, 10, 12,13};
        int []arr2 = {2, 4, 6, 8, 10, 12};
        int n = arr2.Length;

        // Solve is passed
        // both arrays
        Console.Write(findExtra(arr1, arr2, n));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to find an extra
// element present in arr1[]

// Returns index of extra element
// in arr1[]. n is size of arr2[].
// Size of arr1[] is n-1.
function findExtra($arr1, $arr2, $n)
{
    // Initialize result
```

```
    $index = $n;

    // left and right are
    // end points denoting
    // the current range.
    $left = 0; $right = $n - 1;
    while ($left <= $right)
    {
        $mid = ($left+$right) / 2;

        // If middle element is same
        // of both arrays, it means
        // that extra element is after
        // mid so we update left to mid+1
        if ($arr2[$mid] == $arr1[$mid])
            $left = $mid + 1;

        // If middle element is different
        // of the arrays, it means that the
        // index we are searching for is either
        // mid, or before mid. Hence we update
        // right to mid-1.
        else
        {
            $index = $mid;
            $right = $mid - 1;
        }
    }

    // when right is greater than
    // left, our search is complete.
    return $index;
}

// Driver code
{
    $arr1 = array(2, 4, 6, 8,
                  10, 12, 13);
    $arr2 = array(2, 4, 6,
                  8, 10, 12);
    $n = sizeof($arr2) / sizeof($arr2[0]);

    // Solve is passed both arrays
    echo findExtra($arr1, $arr2, $n);
    return 0;
}

// This code is contributed by nitin mittal
```

```
?>
```

Output :

```
6
```

Time complexity : O(log n)

Improved By : parashar, nitin mittal, jit_t

Source

https://www.geeksforgeeks.org/find-index-of-an-extra-element-present-in-one-sorted-array/

Chapter 71

Find index of first occurrence when an unsorted array is sorted

Find index of first occurrence when an unsorted array is sorted - GeeksforGeeks

Given an unsorted array and a number x, find index of first occurrence of x when we sort the array. If x is not present, print -1.

Examples:

```
Input : arr[] = {10, 30, 20, 50, 20}
           x = 20
Output : 1
Sorted array is {10, 20, 20, 30, 50}

Input : arr[] = {10, 30, 20, 50, 20}
           x = 50
Output : -1
50 is not present in array.
```

A **simple solution** is to first sort the array, then dobinary search to find first occurrence.

```
 // CPP program to find index of first
// occurrence of x when array is sorted.
#include<bits/stdc++.h>
using namespace std;

int findFirst(int arr[], int n, int x)
{
```

```
   sort(arr, arr+n);

   // lower_bound returns iterator pointing to
   // first element that does not compare less
   // to x.
   int *ptr = lower_bound(arr, arr+n, x);

   // If x is not present return -1.
   return (*ptr != x)? -1 : (ptr - arr);
}

int main()
{
    int x = 20, arr[] = {10, 30, 20, 50, 20};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << findFirst(arr, n, x);
    return 0;
}
```

Output :

```
1
```

Time Complexity : O(n Log n)

An **efficient solution** is to simply count smaller elements than x.

```
 // CPP program to find index of first
// occurrence of x when array is sorted.
#include<bits/stdc++.h>
using namespace std;

int findFirst(int arr[], int n, int x)
{
   int count = 0;
   bool isX = false;
   for (int i=0; i<n; i++)
   {
      if (arr[i] == x)
          isX = true;
      else if (arr[i] < x)
          count++;
   }
   return (isX == false)? -1 : count;
}

int main()
{
```

```
    int x = 20, arr[] = {10, 30, 20, 50, 20};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << findFirst(arr, n, x);
    return 0;
}
```

Output :

```
1
```

Time Complexity : O(n)

Source

https://www.geeksforgeeks.org/find-index-of-first-occurrence-when-an-unsorted-array-is-sorted/

Chapter 72

Find k closest elements to a given value

Find k closest elements to a given value - GeeksforGeeks

Given a sorted array arr[] and a value X, find the k closest elements to X in arr[].
Examples:

```
Input: K = 4, X = 35
       arr[] = {12, 16, 22, 30, 35, 39, 42,
                45, 48, 50, 53, 55, 56}
Output: 30 39 42 45
```

Note that if the element is present in array, then it should not be in output, only the other closest elements are required.

In the following solutions, it is assumed that all elements of array are distinct.

A **simple solution** is to do linear search for k closest elements.
1) Start from the first element and search for the crossover point (The point before which elements are smaller than or equal to X and after which elements are greater). This step takes O(n) time.
2) Once we find the crossover point, we can compare elements on both sides of crossover point to print k closest elements. This step takes O(k) time.

The time complexity of the above solution is O(n).

An **Optimized Solution** is to find k elements in O(Logn + k) time. The idea is to use Binary Search to find the crossover point. Once we find index of crossover point, we can print k closest elements in O(k) time.

C/C++

```
 #include<stdio.h>

/* Function to find the cross over point (the point before
   which elements are smaller than or equal to x and after
   which greater than x)*/
int findCrossOver(int arr[], int low, int high, int x)
{
  // Base cases
  if (arr[high] <= x) // x is greater than all
    return high;
  if (arr[low] > x)  // x is smaller than all
    return low;

  // Find the middle point
  int mid = (low + high)/2;  /* low + (high - low)/2 */

  /* If x is same as middle element, then return mid */
  if (arr[mid] <= x && arr[mid+1] > x)
    return mid;

  /* If x is greater than arr[mid], then either arr[mid + 1]
    is ceiling of x or ceiling lies in arr[mid+1...high] */
  if(arr[mid] < x)
      return findCrossOver(arr, mid+1, high, x);

  return findCrossOver(arr, low, mid - 1, x);
}

// This function prints k closest elements to x in arr[].
// n is the number of elements in arr[]
void printKclosest(int arr[], int x, int k, int n)
{
    // Find the crossover point
    int l = findCrossOver(arr, 0, n-1, x);
    int r = l+1;   // Right index to search
    int count = 0; // To keep track of count of elements already printed

    // If x is present in arr[], then reduce left index
    // Assumption: all elements in arr[] are distinct
    if (arr[l] == x) l--;

    // Compare elements on left and right of crossover
    // point to find the k closest elements
    while (l >= 0 && r < n && count < k)
    {
        if (x - arr[l] < arr[r] - x)
            printf("%d ", arr[l--]);
        else
```

```
            printf("%d ", arr[r++]);
        count++;
    }

    // If there are no more elements on right side, then
    // print left elements
    while (count < k && l >= 0)
        printf("%d ", arr[l--]), count++;

    // If there are no more elements on left side, then
    // print right elements
    while (count < k && r < n)
        printf("%d ", arr[r++]), count++;
}

/* Driver program to check above functions */
int main()
{
   int arr[] ={12, 16, 22, 30, 35, 39, 42,
               45, 48, 50, 53, 55, 56};
   int n = sizeof(arr)/sizeof(arr[0]);
   int x = 35, k = 4;
   printKclosest(arr, x, 4, n);
   return 0;
}
```

Java

```
 // Java program to find k closest elements to a given value
class KClosest
{
    /* Function to find the cross over point (the point before
       which elements are smaller than or equal to x and after
       which greater than x)*/
    int findCrossOver(int arr[], int low, int high, int x)
    {
        // Base cases
        if (arr[high] <= x) // x is greater than all
            return high;
        if (arr[low] > x)  // x is smaller than all
            return low;

        // Find the middle point
        int mid = (low + high)/2;  /* low + (high - low)/2 */

        /* If x is same as middle element, then return mid */
        if (arr[mid] <= x && arr[mid+1] > x)
            return mid;
```

```
    /* If x is greater than arr[mid], then either arr[mid + 1]
      is ceiling of x or ceiling lies in arr[mid+1...high] */
    if(arr[mid] < x)
        return findCrossOver(arr, mid+1, high, x);

    return findCrossOver(arr, low, mid - 1, x);
}

// This function prints k closest elements to x in arr[].
// n is the number of elements in arr[]
void printKclosest(int arr[], int x, int k, int n)
{
    // Find the crossover point
    int l = findCrossOver(arr, 0, n-1, x);
    int r = l+1;   // Right index to search
    int count = 0; // To keep track of count of elements
                   // already printed

    // If x is present in arr[], then reduce left index
    // Assumption: all elements in arr[] are distinct
    if (arr[l] == x) l--;

    // Compare elements on left and right of crossover
    // point to find the k closest elements
    while (l >= 0 && r < n && count < k)
    {
        if (x - arr[l] < arr[r] - x)
            System.out.print(arr[l--]+" ");
        else
            System.out.print(arr[r++]+" ");
        count++;
    }

    // If there are no more elements on right side, then
    // print left elements
    while (count < k && l >= 0)
    {
        System.out.print(arr[l--]+" ");
        count++;
    }


    // If there are no more elements on left side, then
    // print right elements
    while (count < k && r < n)
    {
        System.out.print(arr[r++]+" ");
```

```
            count++;
        }
    }

    /* Driver program to check above functions */
    public static void main(String args[])
    {
        KClosest ob = new KClosest();
        int arr[] = {12, 16, 22, 30, 35, 39, 42,
                     45, 48, 50, 53, 55, 56
                    };
        int n = arr.length;
        int x = 35, k = 4;
        ob.printKclosest(arr, x, 4, n);
    }
}
/* This code is contributed by Rajat Mishra */
```

C#

```
 // C# program to find k closest elements to
// a given value
using System;

class GFG {

    /* Function to find the cross over point
    (the point before which elements are
    smaller than or equal to x and after which
    greater than x)*/
    static int findCrossOver(int []arr, int low,
                                int high, int x)
    {

        // Base cases
        // x is greater than all
        if (arr[high] <= x)
            return high;

        // x is smaller than all
        if (arr[low] > x)
            return low;

        // Find the middle point
        /* low + (high - low)/2 */
        int mid = (low + high)/2;

        /* If x is same as middle element, then
```

```
    return mid */
    if (arr[mid] <= x && arr[mid+1] > x)
        return mid;

    /* If x is greater than arr[mid], then
    either arr[mid + 1] is ceiling of x or
    ceiling lies in arr[mid+1...high] */
    if(arr[mid] < x)
        return findCrossOver(arr, mid+1,
                                high, x);

    return findCrossOver(arr, low, mid - 1, x);
}

// This function prints k closest elements
// to x in arr[]. n is the number of
// elements in arr[]
static void printKclosest(int []arr, int x,
                            int k, int n)
{

    // Find the crossover point
    int l = findCrossOver(arr, 0, n-1, x);

    // Right index to search
    int r = l + 1;

    // To keep track of count of elements
    int count = 0;

    // If x is present in arr[], then reduce
    // left index Assumption: all elements in
    // arr[] are distinct
    if (arr[l] == x) l--;

    // Compare elements on left and right of
    // crossover point to find the k closest
    // elements
    while (l >= 0 && r < n && count < k)
    {
        if (x - arr[l] < arr[r] - x)
            Console.Write(arr[l--]+" ");
        else
            Console.Write(arr[r++]+" ");
        count++;
    }

    // If there are no more elements on right
```

```
        // side, then print left elements
        while (count < k && l >= 0)
        {
            Console.Write(arr[l--]+" ");
            count++;
        }

        // If there are no more elements on left
        // side, then print right elements
        while (count < k && r < n)
        {
            Console.Write(arr[r++] + " ");
            count++;
        }
    }

    /* Driver program to check above functions */
    public static void Main()
    {
        int []arr = {12, 16, 22, 30, 35, 39, 42,
                        45, 48, 50, 53, 55, 56};
        int n = arr.Length;
        int x = 35;
        printKclosest(arr, x, 4, n);
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP Program to Find k closest
// elements to a given value

/* Function to find the cross
   over point (the point before
   which elements are smaller
   than or equal to x and after
   which greater than x) */
function findCrossOver($arr, $low,
                       $high, $x)
{

    // Base cases

    // x is greater than all
    if ($arr[$high] <= $x)
```

```
        return $high;

    // x is smaller than all
    if ($arr[$low] > $x)
        return $low;

    // Find the middle point
    /* low + (high - low)/2 */
    $mid = ($low + $high)/2;

    /* If x is same as middle
       element, then return mid */
    if ($arr[$mid] <= $x and
        $arr[$mid + 1] > $x)
        return $mid;

    /* If x is greater than arr[mid],
       then either arr[mid + 1] is
       ceiling of x or ceiling lies
       in arr[mid+1...high] */
    if($arr[$mid] < $x)
        return findCrossOver($arr, $mid + 1,
                             $high, $x);

    return findCrossOver($arr, $low,
                      $mid - 1, $x);
}

// This function prints k
// closest elements to x in arr[].
// n is the number of elements
// in arr[]
function printKclosest($arr, $x, $k, $n)
{

    // Find the crossover point
    $l = findCrossOver($arr, 0, $n - 1, $x);

    // Right index to search
    $r = $l + 1;

    // To keep track of count of
    // elements already printed
    $count = 0;

    // If x is present in arr[],
    // then reduce left index
    // Assumption: all elements
```

```
    // in arr[] are distinct
    if ($arr[$l] == $x) $l--;

    // Compare elements on left
    // and right of crossover
    // point to find the k
    // closest elements
    while ($l >= 0 and $r < $n
              and $count < $k)
    {
        if ($x - $arr[$l] < $arr[$r] - $x)
            echo $arr[$l--]," ";
        else
            echo $arr[$r++]," ";
        $count++;
    }

    // If there are no more
    // elements on right side,
    // then print left elements
    while ($count < $k and $l >= 0)
        echo $arr[$l--]," "; $count++;

    // If there are no more
    // elements on left side,
    // then print right elements
    while ($count < $k and $r < $n)
        echo $arr[$r++]; $count++;
}

// Driver Code
$arr =array(12, 16, 22, 30, 35, 39, 42,
                45, 48, 50, 53, 55, 56);
$n = count($arr);
$x = 35; $k = 4;

printKclosest($arr, $x, 4, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
39 30 42 45
```

The time complexity of this method is O(Logn + k).

Exercise: Extend the optimized solution to work for duplicates also, i.e., to work for arrays where elements don't have to be distinct.

This article is contributed by **Rahul Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/find-k-closest-elements-given-value/

Chapter 73

Find k maximum elements of array in original order

Find k maximum elements of array in original order - GeeksforGeeks

Given an array arr[] and an integer k, we need to print k maximum elements of given array. The elements should printed in the order of the input.

Note : k is always less than or equal to n.

Examples:

```
Input : arr[] = {10 50 30 60 15}
        k = 2
Output : 50 60
The top 2 elements are printed
as per their appearance in original
array.

Input : arr[] = {50 8 45 12 25 40 84}
            k = 3
Output : 50 45 84
```

Method 1: We search for the maximum element k times in the given array. Each time we find one maximum element, we print it and repalce it with minus infinite (INT_MIN in C) in the array. The time complexity of this method is O(n*k).

Method 2: In this method, we store the original array in a new array and will sort the new array in descending order. After sorting, we iterate the original array from 0 to n and print all those elements that appear in first k elements of new array. For searching, we can do Binary Search.

C++

```
 // CPP program to find k maximum elements
// of array in original order
#include <bits/stdc++.h>
using namespace std;

// Function to print m Maximum elements
void printMax(int arr[], int k, int n)
{
    // vector to store the copy of the
    // original array
    vector<int> brr(arr, arr + n);

    // Sorting the vector in descending
    // order. Please refer below link for
    // details
    // https://www.geeksforgeeks.org/sort-c-stl/
    sort(brr.begin(), brr.end(), greater<int>());

    // Traversing through original array and
    // printing all those elements that are
    // in first k of sorted vector.
    // Please refer https://goo.gl/44Rwgt
    // for details of binary_search()
    for (int i = 0; i < n; ++i)
        if (binary_search(brr.begin(),
                  brr.begin() + k, arr[i],
                        greater<int>()))
            cout << arr[i] << " ";
}

// Driver code
int main()
{
    int arr[] = { 50, 8, 45, 12, 25, 40, 84 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    printMax(arr, k, n);
    return 0;
}
```

Java

```
 // Java program to find k maximum
// elements of array in original order
import java.util.Arrays;
import java.util.Collections;

public class GfG {
```

```
    // Function to print m Maximum elements
    public static void printMax(int arr[], int k, int n)
    {
        // Array to store the copy
        // of the original array
        Integer[] brr = new Integer[n];

        for (int i = 0; i < n; i++)
        brr[i] = arr[i];

        // Sorting the array in
        // descending order
        Arrays.sort(brr, Collections.reverseOrder());

        // Traversing through original array and
        // printing all those elements that are
        // in first k of sorted array.
        // Please refer https://goo.gl/uj5RCD
        // for details of Arrays.binarySearch()
        for (int i = 0; i < n; ++i)
            if (Arrays.binarySearch(brr, arr[i],
                    Collections.reverseOrder()) >= 0
                 && Arrays.binarySearch(brr, arr[i],
                    Collections.reverseOrder()) < k)

                System.out.print(arr[i]+ " ");
    }

    // Driver code
    public static void main(String args[])
    {
        int arr[] = { 50, 8, 45, 12, 25, 40, 84 };
        int n = arr.length;
        int k = 3;
        printMax(arr, k, n);
    }
}

// This code is contributed by Swetank Modi
```

Output :

```
50 45 84
```

Time Complexity : O(n Log n) for sorting.
Auxiliary Space : O(n)

Source

https://www.geeksforgeeks.org/find-k-maximum-elements-array-original-order/

Chapter 74

Find k-th smallest element in given n ranges

Find k-th smallest element in given n ranges - GeeksforGeeks

Given n and q, i.e, the number of ranges and number of queries, find the kth smallest element for each query (assume k>1).Print the value of kth smallest element if it exists, else print -1.

Examples :

```
Input : arr[] = {{1, 4}, {6, 8}}
        queries[] = {2, 6, 10};
Output : 2
         7
        -1
After combining the given ranges, the numbers
become 1 2 3 4 6 7 8. As here 2nd element is 2,
so we print 2. As 6th element is 7, so we print
7 and as 10th element doesn't exist, so we
print -1.

Input : arr[] = {{2, 6}, {5, 7}}
        queries[] = {5, 8};
Output : 6
        -1
After combining the given ranges, the numbers
become 2 3 4 5 6 7. As here 5th element is 6,
so we print 6 and as 8th element doesn't exist,
so we print -1.
```

The idea is to first Prerequisite : Merge Overlapping Intervals and keep all intervals sorted

in ascending order of start time. After merging in an array merged[], we use linear search to find kth smallest element. Below is the implementation of the above approach :

```
 // C++ implementation to solve k queries
// for given n ranges
#include <bits/stdc++.h>
using namespace std;

// Structure to store the
// start and end point
struct Interval
{
    int s;
    int e;
};

// Comparison function for sorting
bool comp(Interval a, Interval b)
{
    return a.s < b.s;
}

// Function to find Kth smallest number in a vector
// of merged intervals
int kthSmallestNum(vector<Interval> merged, int k)
{
    int n = merged.size();

    // Traverse merged[] to find
    // Kth smallest element using Linear search.
    for (int j = 0; j < n; j++)
    {
        if (k <= abs(merged[j].e -
                     merged[j].s + 1))
            return (merged[j].s + k - 1);

        k = k - abs(merged[j].e -
                     merged[j].s + 1);
    }

    if (k)
        return -1;
}

// To combined both type of ranges,
// overlapping as well as non-overlapping.
void mergeIntervals(vector<Interval> &merged,
                 Interval arr[], int n)
```

```
{
    // Sorting intervals according to start
    // time
    sort(arr, arr + n, comp);

    // Merging all intervals into merged
    merged.push_back(arr[0]);
    for (int i = 1; i < n; i++)
    {
        // To check if starting point of next
        // range is lying between the previous
        // range and ending point of next range
        // is greater than the Ending point
        // of previous range then update ending
        // point of previous range by ending
        // point of next range.
        Interval prev = merged.back();
        Interval curr = arr[i];
        if ((curr.s >= prev.s &&
             curr.s <= prev.e) &&
            (curr.e > prev.e))

            merged.back().e = curr.e;

        else
        {
            // If starting point of next range
            // is greater than the ending point
            // of previous range then store next range
            // in merged[].
            if (curr.s > prev.e)
                merged.push_back(curr);
        }
    }
}

// Driver\'s Function
int main()
{
    Interval arr[] = {{2, 6}, {4, 7}};
    int n = sizeof(arr)/sizeof(arr[0]);
    int query[] = {5, 8};
    int q = sizeof(query)/sizeof(query[0]);

    // Merge all intervals into merged[]
    vector<Interval>merged;
    mergeIntervals(merged, arr, n);
```

```
    // Processing all queries on merged
    // intervals
    for (int i = 0; i < q; i++)
        cout << kthSmallestNum(merged, query[i])
             << endl;

    return 0;
}
```

Output:

```
 6
 -1
```

Time Complexity : O(nlog(n))

Source

https://www.geeksforgeeks.org/find-k-th-smallest-element-in-given-n-ranges/

Chapter 75

Find kth node from Middle towards Head of a Linked List

Find kth node from Middle towards Head of a Linked List - GeeksforGeeks

Given a Linked List and a number K. The task is to print the value of the K-th node from the middle towards the beginning of the List. If no such element exists, then print "-1".

Note: Position of middle node is: (n/2)+1, where n is the total number of nodes in the list.

Examples:

```
Input :  List is 1->2->3->4->5->6->7
         K= 2
Output : 2

Input :  list is 7->8->9->10->11->12
         K = 3
Output : 7
```

Traverse the List from beginning to end and count the total number of nodes. Now, suppose n is the total number of nodes in the List. Therefore, the middle node will be at the position (n/2)+1. Now, the task remains to print the node at $(n/2 + 1 - k)^{th}$ position from the head of the List.

Below is the implementation of the above idea:

```
 // CPP program to find kth node from middle
// towards Head of the Linked List

#include <bits/stdc++.h>

using namespace std;
```

```
// Linked list node
struct Node {
    int data;
    struct Node* next;
};

/* Given a reference (pointer to
   pointer) to the head of a list
   and an int, push a new node on
   the front of the list. */
void push(struct Node** head_ref,
          int new_data)
{
    struct Node* new_node = new Node;
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

// Function to count number of nodes
int getCount(struct Node* head)
{
    int count = 0; // Initialize count
    struct Node* current = head; // Initialize current
    while (current != NULL) {
        count++;
        current = current->next;
    }
    return count;
}

// Function to get the kth node from the mid
// towards begin of the linked list
int printKthfrommid(struct Node* head_ref, int k)
{
    // Get the count of total number of
    // nodes in the linked list
    int n = getCount(head_ref);

    int reqNode = ((n / 2 + 1) - k);

    // If no such node exists, return -1
    if (reqNode <= 0) {
        return -1;
    }

    // Find node at position reqNode
```

```
    else {
        struct Node* current = head_ref;

        // the index of the
        // node we're currently
        // looking at
        int count = 1;
        while (current != NULL) {
            if (count == reqNode)
                return (current->data);
            count++;
            current = current->next;
        }
    }
}

// Driver code
int main()
{
    // start with empty list
    struct Node* head = NULL;
    int k = 2;

    // create linked list
    // 1->2->3->4->5->6->7
    push(&head, 7);
    push(&head, 6);
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);

    cout << printKthfrommid(head, 2);

    return 0;
}
```

Output:

```
2
```

Time Complexity: O(n), where n is the length of the list.
Auxiliary Space: O(1)

Source

https://www.geeksforgeeks.org/find-kth-node-from-middle-towards-head-of-a-linked-list/

Chapter 76

Find last index of a character in a string

Find last index of a character in a string - GeeksforGeeks

Given a string str and a character x, find last index of x in str.

Examples :

```
Input : str = "geeks", x = 'e'
Output : 2
Last index of 'e' in "geeks" is: 2

Input : str = "Hello world!", x = 'o'
Output : 7
Last index of 'o' is: 7
```

Method 1 (Simple : Traverse from left) :
Traverse given string from left to right and keep updating index whenever x matches with current character.

C++

```
 // CPP program to find last index of
// character x in given string.
#include <iostream>
using namespace std;

// Returns last index of x if it is present.
// Else returns -1.
int findLastIndex(string& str, char x)
```

```
{
    int index = -1;
    for (int i = 0; i < str.length(); i++)
        if (str[i] == x)
            index = i;
    return index;
}

// Driver code
int main()
{
    // String in which char is to be found
    string str = "geeksforgeeks";

    // char whose index is to be found
    char x = 'e';
    int index = findLastIndex(str, x);
    if (index == -1)
        cout << "Character not found";
    else
        cout << "Last index is " << index;
    return 0;
}
```

Java

```
 // Java program to find last index
// of character x in given string.
import java.io.*;

class GFG {

// Returns last index of x if
// it is present Else returns -1.
static int findLastIndex(String str, Character x)
{
    int index = -1;
    for (int i = 0; i < str.length(); i++)
        if (str.charAt(i) == x)
            index = i;
    return index;
}

// Driver code
public static void main(String[] args)
{
    // String in which char is to be found
    String str = "geeksforgeeks";
```

```
    // char whose index is to be found
    Character x = 'e';

    int index = findLastIndex(str, x);
    if (index == -1)
        System.out.println("Character not found");
    else
        System.out.println("Last index is " + index);
}
}

/* This code is contributed by Prerna Saini */
```

Python3

```
 # A Python program to find last
# index of character x in given
# string.

# Returns last index of x if it
# is present. Else returns -1.
def findLastIndex(str, x):
    index = -1
    for i in range(0, len(str)):
        if str[i] == x:
            index = i
    return index

# Driver program

# String in which char is to be found
str = "geeksforgeeks"

# char whose index is to be found
x = 'e'

index = findLastIndex(str, x)

if index == -1:
    print("Character not found")
else:
    print('Last index is', index)

# This code is contributed by shrikant13.
```

C#

```
 // C# program to find last index
// of character x in given string.
using System;

class GFG {

    // Returns last index of x if
    // it is present Else returns -1.
    static int findLastIndex(string str, char x)
    {
        int index = -1;
        for (int i = 0; i < str.Length; i++)
            if (str[i] == x)
                index = i;
        return index;
    }

    // Driver code
    public static void Main()
    {
        // String in which char is to be found
        string str = "geeksforgeeks";

        // char whose index is to be found
        char x = 'e';

        int index = findLastIndex(str, x);
        if (index == -1)
            Console.WriteLine("Character not found");
        else
            Console.WriteLine("Last index is " + index);
    }
}

/* This code is contributed by vt_m */
```

PHP

```
 <?php
// PHP program to find last index of
// character x in given string.

// Returns last index of
// x if it is present.
// Else returns -1.
function findLastIndex($str, $x)
{
    $index = -1;
```

```
    for ($i = 0; $i < strlen($str); $i++)
        if ($str[$i] == $x)
            $index = $i;
    return $index;
}

// Driver code
// String in which
// char is to be found
$str = "geeksforgeeks";

// char whose index
// is to be found
$x = 'e';
$index = findLastIndex($str, $x);
if ($index == -1)
    echo("Character not found");
else
    echo("Last index is " . $index);

// This code is contributed by Ajit.
?>
```

Output:

```
Last index is 10
```

Time Complexity : $\Theta(n)$

Method 2 (Efficient : Traverse from right) :
In above method 1, we always traverse complete string. In this method, we can avoid complete traversal in all those cases when x is present. The idea is to traverse from right side and stop as soon as we find character.

CPP

```
 // Simple CPP program to find last index of
// character x in given string.
#include <iostream>
using namespace std;

// Returns last index of x if it is present.
// Else returns -1.
int findLastIndex(string& str, char x)
{
    // Traverse from right
    for (int i = str.length() - 1; i >= 0; i--)
        if (str[i] == x)
```

```
            return i;

    return -1;
}

// Driver code
int main()
{
    string str = "geeksforgeeks";
    char x = 'e';
    int index = findLastIndex(str, x);
    if (index == -1)
        cout << "Character not found";
    else
        cout << "Last index is " << index;
    return 0;
}
```

Java

```
 // Java code to find last index
// character x in given string.
import java.io.*;
class GFG {

// Returns last index of x if
// it is present. Else returns -1.
static int findLastIndex(String str, Character x)
{
    // Traverse from right
    for (int i = str.length() - 1; i >= 0; i--)
        if (str.charAt(i) == x)
            return i;

    return -1;
}

// Driver code
public static void main(String[] args)
{
    String str = "geeksforgeeks";
    Character x = 'e';
    int index = findLastIndex(str, x);
    if (index == -1)
        System.out.println("Character not found");
    else
        System.out.println("Last index is " + index);
}
```

```
}
// This code is contributed by Prerna Saini
```

Python3

```
 # Simple Python3 program to find last
# index of character x in given string.

# Returns last index of x if it is
# present. Else returns -1.
def findLastIndex(str, x):

    # Traverse from right
    for i in range(len(str) - 1, -1,-1):
        if (str[i] == x):
            return i

    return -1

# Driver code
str = "geeksforgeeks"
x = 'e'
index = findLastIndex(str, x)

if (index == -1):
    print("Character not found")
else:
    print("Last index is " ,index)

# This code is contributed by Smitha
```

C#

```
 // C# code to find last index
// character x in given string.
using System;

class GFG {

    // Returns last index of x if
    // it is present. Else returns -1.
    static int findLastIndex(string str, char x)
    {
        // Traverse from right
        for (int i = str.Length - 1; i >= 0; i--)
            if (str[i] == x)
                return i;
```

```
        return -1;
    }

    // Driver code
    public static void Main()
    {
        string str = "geeksforgeeks";
        char x = 'e';
        int index = findLastIndex(str, x);
        if (index == -1)
            Console.WriteLine("Character not found");
        else
            Console.WriteLine("Last index is " + index);
    }
}
// This code is contributed by vt_m
```

PHP

```
 <?php
// Simple PHP program to find last index
// of character x in given string.

// Returns last index of x if it
// is present. Else returns -1.
function findLastIndex($str, $x)
{

    // Traverse from right
    for ($i = strlen($str) - 1; $i >= 0; $i--)
        if ($str[$i] == $x)
            return $i;

    return -1;
}

// Driver code
$str = "geeksforgeeks";
$x = 'e';
$index = findLastIndex($str, $x);
if ($index == -1)
    echo("Character not found");
else
    echo("Last index is " . $index);

// This code is contributed by Ajit.
?>
```

Output:

```
Last index is 10
```

Time Complexity : O(n)

Improved By : shrikanth13, jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-last-index-character-string/

Chapter 77

Find maximum element of each row in a matrix

Find maximum element of each row in a matrix - GeeksforGeeks

Given a matrix, the task is to find the maximum element of each row.

Examples :

```
Input :  [1, 2, 3]
         [1, 4, 9]
         [76, 34, 21]

Output :
3
9
76

Input : [1, 2, 3, 21]
        [12, 1, 65, 9]
        [1, 56, 34, 2]
Output :
21
65
56
```

Approach : Approach is very simple. The idea is to run the loop for no_of_rows. Check each element inside the row and find for the maximum element. Finally, print the element.

Below is the implementation :

```
# Python program to find maximum
```

```
# element of each row in a matrix

# importing numpy
import numpy

# Function to get max element
def maxelement(arr):

    # get number of rows and columns
    no_of_rows = len(arr)
    no_of_column = len(arr[0])

    for i in range(no_of_rows):

        # Initialize max1 to 0 at begining
        # of finding max element of each row
        max1 = 0
        for j in range(no_of_column):
            if arr[i][j] > max1 :
                max1 = arr[i][j]

        # print maximum element of each row
        print(max1)

# Driver Code
arr = [[3, 4, 1, 8],
       [1, 4, 9, 11],
       [76, 34, 21, 1],
       [2, 1, 4, 5]]

# Calling the function
maxelement(arr)
```

Output :

```
8
11
76
5
```

Source

https://www.geeksforgeeks.org/find-maximum-element-row-matrix/

Chapter 78

Find next greater number with same set of digits

Find next greater number with same set of digits - GeeksforGeeks

Given a number n, find the smallest number that has same set of digits as n and is greater than n. If x is the greatest possible number with its set of digits, then print “not possible”.

Examples:
For simplicity of implementation, we have considered input number as a string.

```
Input:  n = "218765"
Output: "251678"

Input:  n = "1234"
Output: "1243"

Input: n = "4321"
Output: "Not Possible"

Input: n = "534976"
Output: "536479"
```

Following are few observations about the next greater number.
1) If all digits sorted in descending order, then output is always “Not Possible”. For example, 4321.
2) If all digits are sorted in ascending order, then we need to swap last two digits. For example, 1234.
3) For other cases, we need to process the number from rightmost side (why? because we need to find the smallest of all greater numbers)

You can now try developing an algorithm yourself.

Following is the algorithm for finding the next greater number.
I) Traverse the given number from rightmost digit, keep traversing till you find a digit which is smaller than the previously traversed digit. For example, if the input number is "534976", we stop at **4** because 4 is smaller than next digit 9. If we do not find such a digit, then output is "Not Possible".

II) Now search the right side of above found digit 'd' for the smallest digit greater than 'd'. For "53**4**976 , the right side of 4 contains "976". The smallest digit greater than 4 is **6**.

III) Swap the above found two digits, we get 53**6**974 in above example.

IV) Now sort all digits from position next to 'd' to the end of number. The number that we get after sorting is the output. For above example, we sort digits in bold 536**974**. We get "536**479**" which is the next greater number for input 534976.

Following is C++ implementation of above approach.

C

```
 // C++ program to find the smallest number which greater than a given number
// and has same set of digits as given number
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

// Utility function to swap two digits
void swap(char *a, char *b)
{
    char temp = *a;
    *a = *b;
    *b = temp;
}

// Given a number as a char array number[], this function finds the
// next greater number.  It modifies the same array to store the result
void findNext(char number[], int n)
{
    int i, j;

    // I) Start from the right most digit and find the first digit that is
    // smaller than the digit next to it.
    for (i = n-1; i > 0; i--)
        if (number[i] > number[i-1])
           break;

    // If no such digit is found, then all digits are in descending order
    // means there cannot be a greater number with same set of digits
    if (i==0)
    {
```

```
        cout << "Next number is not possible";
        return;
    }

    // II) Find the smallest digit on right side of (i-1)'th digit that is
    // greater than number[i-1]
    int x = number[i-1], smallest = i;
    for (j = i+1; j < n; j++)
        if (number[j] > x && number[j] < number[smallest])
            smallest = j;

    // III) Swap the above found smallest digit with number[i-1]
    swap(&number[smallest], &number[i-1]);

    // IV) Sort the digits after (i-1) in ascending order
    sort(number + i, number + n);

    cout << "Next number with same set of digits is " << number;

    return;
}

// Driver program to test above function
int main()
{
    char digits[] = "534976";
    int n = strlen(digits);
    findNext(digits, n);
    return 0;
}
```

Java

```
 // Java program to find next greater
// number with same set of digits.
import java.util.Arrays;

public class nextGreater
{
    // Utility function to swap two digit
    static void swap(char ar[], int i, int j)
    {
        char temp = ar[i];
        ar[i] = ar[j];
        ar[j] = temp;
    }

    // Given a number as a char array number[],
```

```
// this function finds the next greater number.
// It modifies the same array to store the result
static void findNext(char ar[], int n)
{
    int i;

    // I) Start from the right most digit
    // and find the first digit that is smaller
    // than the digit next to it.
    for (i = n - 1; i > 0; i--)
    {
        if (ar[i] > ar[i - 1]) {
            break;
        }
    }

    // If no such digit is found, then all
    // digits are in descending order means
    // there cannot be a greater number with
    // same set of digits
    if (i == 0)
    {
        System.out.println("Not possible");
    }
    else
    {
        int x = ar[i - 1], min = i;

        // II) Find the smallest digit on right
        // side of (i-1)'th digit that is greater
        // than number[i-1]
        for (int j = i + 1; j < n; j++)
        {
            if (ar[j] > x && ar[j] < ar[min])
            {
                min = j;
            }
        }

        // III) Swap the above found smallest
        // digit with number[i-1]
        swap(ar, i - 1, min);

        // IV) Sort the digits after (i-1)
        // in ascending order
        Arrays.sort(ar, i, n);
        System.out.print("Next number with same" +
                                 " set of digits is ");
```

```
            for (i = 0; i < n; i++)
                System.out.print(ar[i]);
        }
    }

    public static void main(String[] args)
    {
        char digits[] = { '5','3','4','9','7','6' };
        int n = digits.length;
        findNext(digits, n);
    }
}
```

Python

```
 # Python program to find the smallest number which
# is greater than a given no. has same set of
# digits as given number

# Given number as int array, this function finds the
# greatest number and returns the number as integer
def findNext(number,n):

     # Start from the right most digit and find the first
     # digit that is smaller than the digit next to it
     for i in range(n-1,0,-1):
         if number[i] > number[i-1]:
             break

     # If no such digit found,then all numbers are in
     # descending order, no greater number is possible
     if i == 0:
         print "Next number not possible"
         return

     # Find the smallest digit on the right side of
     # (i-1)'th digit that is greater than number[i-1]
     x = number[i-1]
     smallest = i
     for j in range(i+1,n):
         if number[j] > x and number[j] < number[smallest]:
             smallest = j

     # Swapping the above found smallest digit with (i-1)'th
     number[smallest],number[i-1] = number[i-1], number[smallest]

     # X is the final number, in integer datatype
     x = 0
```

```
    # Converting list upto i-1 into number
    for j in range(i):
        x = x * 10 + number[j]

    # Sort the digits after i-1 in ascending order
    number = sorted(number[i:])
    # converting the remaining sorted digits into number
    for j in range(n-i):
        x = x * 10 + number[j]

    print "Next number with set of digits is",x


# Driver Program to test above function
digits = "534976"

# converting into integer array,
# number becomes [5,3,4,9,7,6]
number = map(int ,digits)
findNext(number, len(digits))

# This code is contributed by Harshit Agrawal
```

Output:

```
Next number with same set of digits is 536479
```

The above implementation can be optimized in following ways.
1) We can use binary search in step II instead of linear search.
2) In step IV, instead of doing simple sort, we can apply some clever technique to do it in linear time. Hint: We know that all digits are linearly sorted in reverse order except one digit which was swapped.

With above optimizations, we can say that the time complexity of this method is O(n).

This article is contributed by **Rahul Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/find-next-greater-number-set-digits/

Chapter 79

Find number of pairs in an array such that their XOR is 0

Find number of pairs in an array such that their XOR is 0 - GeeksforGeeks

Given an array $A[]$ of size N. Find the number of pairs (i, j) such that A_i XOR A_j = 0, and 1 <= i < j <= N.

Examples :

```
Input : A[] = {1, 3, 4, 1, 4}
Output : 2
Explanation : Index (0, 3) and (2, 4)

Input : A[] = {2, 2, 2}
Output : 3
```

First Approach : **Sorting**

A_i XOR A_j = 0 is only satisfied when $A_i = A_j$. Therefore, we will first sort the array and then count the frequency of each element. By combinatorics, we can observe that if frequency of some element is $count$ then, it will contribute $count * (count - 1) / 2$ to the answer.

Below is the implementation of above approach :

C++

```
// C++ program to find number
```

```
// of pairs in an array such
// that their XOR is 0
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the
// count
int calculate(int a[], int n)
{
    // Sorting the list using
    // built in function
    sort(a, a + n);

    int count = 1;
    int answer = 0;

    // Traversing through the
    // elements
    for (int i = 1; i < n; i++) {

        if (a[i] == a[i - 1]){

            // Counting frequency of each
            // elements
            count += 1;

        }
        else
        {
            // Adding the contribution of
            // the frequency to the answer
            answer = answer + (count * (count - 1)) / 2;
            count = 1;
        }
    }

    answer = answer + (count * (count - 1)) / 2;

    return answer;
}

// Driver Code
int main()
{

    int a[] = { 1, 2, 1, 2, 4 };
    int n = sizeof(a) / sizeof(a[0]);
```

```
    // Print the count
    cout << calculate(a, n);
    return 0;
}

// This article is contributed by Sahil_Bansall.
```

Java

```
 // Java program to find number
// of pairs in an array such
// that their XOR is 0
import java.util.*;

class GFG
{
    // Function to calculate
    // the count
    static int calculate(int a[], int n)
    {
        // Sorting the list using
        // built in function
        Arrays.sort(a);

        int count = 1;
        int answer = 0;

        // Traversing through the
        // elements
        for (int i = 1; i < n; i++)
        {

            if (a[i] == a[i - 1])
            {
                // Counting frequency of each
                // elements
                count += 1;

            }
            else
            {
                // Adding the contribution of
                // the frequency to the answer
                answer = answer + (count * (count - 1)) / 2;
                count = 1;
            }
        }
```

```
        answer = answer + (count * (count - 1)) / 2;

        return answer;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int a[] = { 1, 2, 1, 2, 4 };
        int n = a.length;

        // Print the count
        System.out.println(calculate(a, n));
    }
}

// This code is contributed by Ansu Kumari.
```

Python3

```
 # Python3 program to find number of pairs
# in an array such that their XOR is 0

# Function to calculate the count
def calculate(a) :

    # Sorting the list using
    # built in function
    a.sort()

    count = 1
    answer = 0

    # Traversing through the elements
    for i in range(1, len(a)) :

        if a[i] == a[i - 1] :

            # Counting frequncy of each elements
            count += 1

        else :

            # Adding the contribution of
            # the frequency to the answer
            answer = answer + count * (count - 1) // 2
            count = 1
```

```
    answer = answer + count * (count - 1) // 2

    return answer


# Driver Code
if __name__ == '__main__':

    a = [1, 2, 1, 2, 4]

    # Print the count
    print(calculate(a))
```

C#

```
 // Java program to find number
// of pairs in an array such
// that their XOR is 0
using System;

class GFG
{
    // Function to calculate
    // the count
    static int calculate(int []a, int n)
    {
        // Sorting the list using
        // built in function
        Array.Sort(a);

        int count = 1;
        int answer = 0;

        // Traversing through the
        // elements
        for (int i = 1; i < n; i++)
        {

            if (a[i] == a[i - 1])
            {
                // Counting frequency of each
                // elements
                count += 1;

            }
            else
            {
                // Adding the contribution of
```

```
                // the frequency to the answer
                answer = answer + (count * (count - 1)) / 2;
                count = 1;
            }
        }

        answer = answer + (count * (count - 1)) / 2;

        return answer;
    }

    // Driver Code
    public static void Main ()
    {
        int []a = { 1, 2, 1, 2, 4 };
        int n = a.Length;

        // Print the count
        Console.WriteLine(calculate(a, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find number
// of pairs in an array such
// that their XOR is 0

// Function to calculate
// the count
function calculate($a, $n)
{

    // Sorting the list using
    // built in function
    sort($a);

    $count = 1;
    $answer = 0;

    // Traversing through the
    // elements
    for ($i = 1; $i < $n; $i++)
    {
```

```
        if ($a[$i] == $a[$i - 1])
        {

            // Counting frequency of
            // each elements
            $count += 1;

        }

        else
        {

            // Adding the contribution of
            // the frequency to the answer
            $answer = $answer + ($count *
                       ($count - 1)) / 2;
            $count = 1;
        }
    }

    $answer = $answer + ($count *
               ($count - 1)) / 2;

    return $answer;
}

    // Driver Code
    $a = array(1, 2, 1, 2, 4);
    $n = count($a);

    // Print the count
    echo calculate($a, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
2
```

Time Complexity : O(N Log N)

Second Approach : **Hashing (Index Mapping)**

Solution is handy, if we can count the frequency of each element in the array. Index mapping technique can be used to count the frequency of each element.

Below is the implementation of above approach :

C++

```
 // C++ program to find number of pairs
// in an array such that their XOR is 0
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the answer
int calculate(int a[]){

    // Finding the maximum of the array
    int *maximum = max_element(a, a + 5);

    // Creating frequency array
    // With initial value 0
    int frequency[*maximum + 1] = {0};

    // Traversing through the array
    for(int i = 0; i < (*maximum)+1; i++)
    {
        // Counting frequency
        frequency[a[i]] += 1;
    }
    int answer = 0;

    // Traversing through the frequency array
    for(int i = 0; i < (*maximum)+1; i++)
    {
        // Calculating answer
        answer = answer + frequency[i] * (frequency[i] - 1) ;
    }
    return answer/2;
}

// Driver Code
int main()
{
   int a[] = {1, 2, 1, 2, 4};

   // Function calling
   cout << (calculate(a));
}

// This code is contributed by Smitha
```

Python 3

```
 # Python3 program to find number of pairs
# in an array such that their XOR is 0

# Function to calculate the answer
def calculate(a) :

    # Finding the maximum of the array
    maximum = max(a)

    # Creating frequency array
    # With initial value 0
    frequency = [0 for x in range(maximum + 1)]

    # Traversing through the array
    for i in a :

        # Counting frequency
        frequency[i] += 1

    answer = 0

    # Traversing through the frequency array
    for i in frequency :

        # Calculating answer
        answer = answer + i * (i - 1) // 2

    return answer

# Driver Code
a = [1, 2, 1, 2, 4]
print(calculate(a))
```

PHP

Output :

```
2
```

Time Complexity : O(N)

Note : Index Mapping method can only be used when the numbers in the array are not large. In such cases, sorting method can be used.

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-number-pairs-array-xor-0/

Chapter 80

Find position of an element in a sorted array of infinite numbers

Find position of an element in a sorted array of infinite numbers - GeeksforGeeks

Suppose you have a sorted array of infinite numbers, how would you search an element in the array?

Source: Amazon Interview Experience.

Since array is sorted, the first thing clicks into mind is binary search, but the problem here is that we don't know size of array.
If the array is infinite, that means we don't have proper bounds to apply binary search. So in order to find position of key, first we find bounds and then apply binary search algorithm.

Let low be pointing to 1st element and high pointing to 2nd element of array, Now compare key with high index element,
->if it is greater than high index element then copy high index in low index and double the high index.
->if it is smaller, then apply binary search on high and low indices found.

Below are implementations of above algorithm

C++

```
 // C++ program to demonstrate working of an algorithm that finds
// an element in an array of infinite size
#include<iostream>
using namespace std;

// Simple binary search algorithm
int binarySearch(int arr[], int l, int r, int x)
{
    if (r>=l)
    {
```

```
        int mid = l + (r - l)/2;
        if (arr[mid] == x)
            return mid;
        if (arr[mid] > x)
            return binarySearch(arr, l, mid-1, x);
        return binarySearch(arr, mid+1, r, x);
    }
    return -1;
}

// function takes an infinite size array and a key to be
//  searched and returns its position if found else -1.
// We don't know size of arr[] and we can assume size to be
// infinite in this function.
// NOTE THAT THIS FUNCTION ASSUMES arr[] TO BE OF INFINITE SIZE
// THEREFORE, THERE IS NO INDEX OUT OF BOUND CHECKING
int findPos(int arr[], int key)
{
    int l = 0, h = 1;
    int val = arr[0];

    // Find h to do binary search
    while (val < key)
    {
        l = h;         // store previous high
        h = 2*h;       // double high index
        val = arr[h]; // update new val
    }

    // at this point we have updated low and
    // high indices, Thus use binary search
    // between them
    return binarySearch(arr, l, h, key);
}

// Driver program
int main()
{
    int arr[] = {3, 5, 7, 9, 10, 90, 100, 130,
                                 140, 160, 170};
    int ans = findPos(arr, 10);
    if (ans==-1)
        cout << "Element not found";
    else
        cout << "Element found at index " << ans;
    return 0;
}
```

Java

```
 // Java program to demonstrate working of
// an algorithm that finds an element in an
// array of infinite size

class Test
{
    // Simple binary search algorithm
    static int binarySearch(int arr[], int l, int r, int x)
    {
        if (r>=l)
        {
            int mid = l + (r - l)/2;
            if (arr[mid] == x)
                return mid;
            if (arr[mid] > x)
                return binarySearch(arr, l, mid-1, x);
            return binarySearch(arr, mid+1, r, x);
        }
        return -1;
    }

    // Method takes an infinite size array and a key to be
    //  searched and returns its position if found else -1.
    // We don't know size of arr[] and we can assume size to be
    // infinite in this function.
    // NOTE THAT THIS FUNCTION ASSUMES arr[] TO BE OF INFINITE SIZE
    // THEREFORE, THERE IS NO INDEX OUT OF BOUND CHECKING
    static int findPos(int arr[],int key)
    {
        int l = 0, h = 1;
        int val = arr[0];

        // Find h to do binary search
        while (val < key)
        {
            l = h;         // store previous high
            h = 2*h;       // double high index
            val = arr[h]; // update new val
        }

        // at this point we have updated low
        //  and high indices, thus use binary
        // search between them
        return binarySearch(arr, l, h, key);
    }
```

```
    // Driver method to test the above function
    public static void main(String[] args)
    {
        int arr[] = new int[]{3, 5, 7, 9, 10, 90,
                              100, 130, 140, 160, 170};
        int ans = findPos(arr,10);

        if (ans==-1)
            System.out.println("Element not found");
        else
            System.out.println("Element found at index " + ans);
    }
}
```

Python

```
 # Python Program to demonstrate working of an algorithm that finds
# an element in an array of infinite size

# Binary search algorithm implementation
def binary_search(arr,l,r,x):

    if r >= l:
        mid = l+(r-l)/2

        if arr[mid] == x:
            return mid

        if arr[mid] > x:
            return binary_search(arr,l,mid-1,x)

        return binary_search(arr,mid+1,r,x)

    return -1

# function takes an infinite size array and a key to be
# searched and returns its position if found else -1.
# We don't know size of a[] and we can assume size to be
# infinite in this function.
# NOTE THAT THIS FUNCTION ASSUMES a[] TO BE OF INFINITE SIZE
# THEREFORE, THERE IS NO INDEX OUT OF BOUND CHECKING
def findPos(a, key):

    l, h, val = 0, 1, arr[0]

    # Find h to do binary search
    while val < key:
        l = h            #store previous high
```

```
        h = 2*h          #double high index
        val = arr[h]       #update new val

    # at this point we have updated low and high indices,
    # thus use binary search between them
    return binary_search(a, l, h, key)

# Driver function
arr = [3, 5, 7, 9, 10, 90, 100, 130, 140, 160, 170]
ans = findPos(arr,10)
if ans == -1:
    print "Element not found"
else:
    print"Element found at index",ans

# This code is contributed by __Devesh Agrawal__
```

C#

```
 // C# program to demonstrate working of an
// algorithm that finds an element in an
// array of infinite size
using System;

class GFG {

    // Simple binary search algorithm
    static int binarySearch(int []arr, int l,
                                int r, int x)
    {
        if (r >= l)
        {
            int mid = l + (r - l)/2;

            if (arr[mid] == x)
                return mid;
            if (arr[mid] > x)
                return binarySearch(arr, l,
                                    mid-1, x);

            return binarySearch(arr, mid+1,
                                       r, x);
        }

        return -1;
    }

    // Method takes an infinite size array
```

```
    // and a key to be searched and returns
    // its position if found else -1. We
    // don't know size of arr[] and we can
    // assume size to be infinite in this
    // function.
    // NOTE THAT THIS FUNCTION ASSUMES
    // arr[] TO BE OF INFINITE SIZE
    // THEREFORE, THERE IS NO INDEX OUT
    // OF BOUND CHECKING
    static int findPos(int []arr,int key)
    {
        int l = 0, h = 1;
        int val = arr[0];

        // Find h to do binary search
        while (val < key)
        {
            l = h;     // store previous high
            h = 2 * h;// double high index
            val = arr[h]; // update new val
        }

        // at this point we have updated low
        // and high indices, thus use binary
        // search between them
        return binarySearch(arr, l, h, key);
    }

    // Driver method to test the above
    // function
    public static void Main()
    {
        int []arr = new int[]{3, 5, 7, 9,
            10, 90, 100, 130, 140, 160, 170};
        int ans = findPos(arr, 10);

        if (ans == -1)
            Console.Write("Element not found");
        else
            Console.Write("Element found at "
                          + "index " + ans);
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to demonstrate working
// of an algorithm that finds an
// element in an array of infinite size

// Simple binary search algorithm
function binarySearch($arr, $l,
                      $r, $x)
{
    if ($r >= $l)
    {
        $mid = $l + ($r - $l)/2;
        if ($arr[$mid] == $x)
            return $mid;
        if ($arr[$mid] > $x)
            return binarySearch($arr, $l,
                                $mid - 1, $x);
        return binarySearch($arr, $mid + 1,
                                    $r, $x);
    }
    return -1;
}

// function takes an infinite
// size array and a key to be
// searched and returns its
// position if found else -1.
// We don't know size of arr[]
// and we can assume size to be
// infinite in this function.
// NOTE THAT THIS FUNCTION ASSUMES
// arr[] TO BE OF INFINITE SIZE
// THEREFORE, THERE IS NO INDEX
// OUT OF BOUND CHECKING
function findPos( $arr, $key)
{
    $l = 0; $h = 1;
    $val = $arr[0];

    // Find h to do binary search
    while ($val < $key)
    {

        // store previous high
        $l = $h;

         // double high index
        $h = 2 * $h;
```

```
        // update new val
        $val = $arr[$h];
    }

    // at this point we have
    // updated low and high
    // indices, Thus use binary
    // search between them
    return binarySearch($arr, $l,
                        $h, $key);
}

    // Driver Code
    $arr = array(3, 5, 7, 9, 10, 90, 100,
                 130, 140, 160, 170);
    $ans = findPos($arr, 10);
    if ($ans==-1)
        echo "Element not found";
    else
        echo "Element found at index " , $ans;

// This code is contributed by anuj_67.
?>
```

Output:

```
Element found at index 4
```

Let p be the position of element to be searched. Number of steps for finding high index 'h' is O(Log p). The value of 'h' must be less than 2*p. The number of elements between h/2 and h must be O(p). Therefore, time complexity of Binary Search step is also O(Log p) and overall time complexity is 2*O(Log p) which is O(Log p).

This article is contributed by **Gaurav Sharma**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/find-position-element-sorted-array-infinite-numbers/

Chapter 81

Find relative complement of two sorted arrays

Find relative complement of two sorted arrays - GeeksforGeeks

Given two sorted arrays arr1 and arr2 of size m and n respectively. We need to find relative complement of two array i.e, arr1 – arr2 which means that we need to find all those elements which are present in arr1 but not in arr2.

Examples:

```
Input : arr1[] = {3, 6, 10, 12, 15}
        arr2[] = {1, 3, 5, 10, 16}
Output : 6 12 15
The elements 6, 12 and 15 are present
in arr[], but not present in arr2[]

Input : arr1[] = {10, 20, 36, 59}
        arr2[] = {5, 10, 15, 59}
Output : 20 36
```

1. Take two pointers i and j which traverse through arr1 and arr2 respectively.
2. If arr1[i] element is smaller than arr2[j] element print this element and increment i.
3. If arr1 element is greater than arr2[j] element then increment j.
4. otherwise increment i and j.

C++

```
 // CPP program to find all those
// elements of arr1[] that are not
// present in arr2[]
```

```
#include <iostream>
using namespace std;

void relativeComplement(int arr1[], int arr2[],
                                int n, int m) {

  int i = 0, j = 0;
  while (i < n && j < m) {

    // If current element in arr2[] is
    // greater, then arr1[i] can't be
    // present in arr2[j..m-1]
    if (arr1[i] < arr2[j]) {
      cout << arr1[i] << " ";
      i++;

    // Skipping smaller elements of
    // arr2[]
    } else if (arr1[i] > arr2[j]) {
      j++;

    // Equal elements found (skipping
    // in both arrays)
    } else if (arr1[i] == arr2[j]) {
      i++;
      j++;
    }
  }

  // Printing remaining elements of
  // arr1[]
  while (i < n)
    cout << arr1[i] << " ";
}

// Driver code
int main() {
  int arr1[] = {3, 6, 10, 12, 15};
  int arr2[] = {1, 3, 5, 10, 16};
  int n = sizeof(arr1) / sizeof(arr1[0]);
  int m = sizeof(arr2) / sizeof(arr2[0]);
  relativeComplement(arr1, arr2, n, m);
  return 0;
}
```

Java

```
 // Java program to find all those
```

```
// elements of arr1[] that are not
// present in arr2[]

class GFG
{
    static void relativeComplement(int arr1[], int arr2[],
                                                int n, int m)
    {

        int i = 0, j = 0;
        while (i < n && j < m)
        {

            // If current element in arr2[] is
            // greater, then arr1[i] can't be
            // present in arr2[j..m-1]
            if (arr1[i] < arr2[j])
            {
                System.out.print(arr1[i] + " ");
                i++;

            // Skipping smaller elements of
            // arr2[]
            } else if (arr1[i] > arr2[j])
            {
                j++;

            // Equal elements found (skipping
            // in both arrays)
            }
            else if (arr1[i] == arr2[j])
            {
                i++;
                j++;
            }
        }

        // Printing remaining elements of
        // arr1[]
        while (i < n)
            System.out.print(arr1[i] + " ");
    }

    // Driver code
    public static void main (String[] args)
    {
        int arr1[] = {3, 6, 10, 12, 15};
        int arr2[] = {1, 3, 5, 10, 16};
```

```
        int n = arr1.length;
        int m = arr2.length;
        relativeComplement(arr1, arr2, n, m);
     }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python program to find all those
# elements of arr1[] that are not
# present in arr2[]

def relativeComplement(arr1, arr2, n, m):

    i = 0
    j = 0
    while (i < n and j < m):

        # If current element in arr2[] is
        # greater, then arr1[i] can't be
        # present in arr2[j..m-1]
        if (arr1[i] < arr2[j]):
            print(arr1[i] , " ", end="")
            i += 1

            # Skipping smaller elements of
            # arr2[]
        elif (arr1[i] > arr2[j]):
            j += 1

            # Equal elements found (skipping
            # in both arrays)
        elif (arr1[i] == arr2[j]):
            i += 1
            j += 1

    # Printing remaining elements of
    # arr1[]
    while (i < n):
        print(arr1[i] , " ", end="")

# Driver code
arr1= [3, 6, 10, 12, 15]
arr2 = [1, 3, 5, 10, 16]
n = len(arr1)
m = len(arr2)
```

```
relativeComplement(arr1, arr2, n, m)

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to find all those
// elements of arr1[] that are not
// present in arr2[]
using System;

namespace Complement
{
    public class GFG
    {

        static void relativeComplement(int []arr1, int []arr2,
                                                    int n, int m)
        {

        int i = 0, j = 0;
        while (i < n && j < m)
        {

            // If current element in arr2[] is
            // greater, then arr1[i] can't be
            // present in arr2[j..m-1]
            if (arr1[i] < arr2[j])
            {
                Console.Write(arr1[i] + " ");
                i++;

            // Skipping smaller elements of
            // arr2[]
            } else if (arr1[i] > arr2[j])
            {
                j++;

            // Equal elements found (skipping
            // in both arrays)
            }
            else if (arr1[i] == arr2[j])
            {
                i++;
                j++;
            }
        }
```

```
        // Printing remaining elements of
        // arr1[]
        while (i < n)
            Console.Write(arr1[i] + " ");
    }

    // Driver code
    public static void Main()
    {
        int []arr1 = {3, 6, 10, 12, 15};
        int []arr2 = {1, 3, 5, 10, 16};
        int n = arr1.Length;
        int m = arr2.Length;
        relativeComplement(arr1,arr2, n, m);
    }
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to find all those
// elements of arr1[] that are not
// present in arr2[]

function relativeComplement($arr1, $arr2,
                                $n, $m)
{

    $i = 0; $j = 0;
    while ($i < $n && $j < $m)
    {

        // If current element in arr2[] is
        // greater, then arr1[i] can't be
        // present in arr2[j..m-1]
        if ($arr1[$i] < $arr2[$j])
        {
            echo $arr1[$i] , " ";
            $i++;

            // Skipping smaller elements of
            // arr2[]
        }
        else if ($arr1[$i] > $arr2[$j])
```

```
        {
            $j++;

            // Equal elements found (skipping
            // in both arrays)
        }
        else if ($arr1[$i] == $arr2[$j])
        {
            $i++;
            $j++;
        }
    }

    // Printing remaining elements of
    // arr1[]
    while ($i < $n)
        echo $arr1[$i] , " ";
}

// Driver code
{
    $arr1 = array(3, 6, 10, 12, 15);
    $arr2 = array(1, 3, 5, 10, 16);
    $n = sizeof($arr1) / sizeof($arr1[0]);
    $m = sizeof($arr2) / sizeof($arr2[0]);
    relativeComplement($arr1, $arr2, $n, $m);
    return 0;
}

// This code is contributed by nitin mittal
?>
```

Output :

```
6 12 15
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/find-relative-complement-two-sorted-arrays/

Chapter 82

Find row number of a binary matrix having maximum number of 1s

Find row number of a binary matrix having maximum number of 1s - GeeksforGeeks

Given a binary matrix (containing only 0 and 1) of order n*n. All rows are sorted already, We need to find the row number with maximum number of 1s. Also find number of 1 in that row.
Note: in case of tie, print smaller row number.

Examples :

```
Input : mat[3][3] = {0, 0, 1,
                     0, 1, 1,
                     0, 0, 0}
Output : Row number = 2, MaxCount = 2

Input : mat[3][3] = {1, 1, 1,
                     1, 1, 1,
                     0, 0, 0}
Output : Row number = 1, MaxCount = 3
```

Basic Approach : Traverse whole of matrix and for each row find the number of 1 and among all that keep updating the row number with maximum number of 1. This approach will result in O(n^2) time complexity.

Better Approach : We can perform a better if we try to apply binary search for finding position of first 1 in each row and as per that we can find the number of 1 from each row as each row is in sorted order. This will result in O(nlogn) time complexity.

Efficient Approach : Start with top left corner with index (1, n) and try to go left until you reach last 1 in that row (jth column), now if we traverse left to that row, we will find 0, so switch to the row just below, with same column. Now your position will be (2, j) again in 2nd row if jth element if 1 try to go left until you find last 1 otherwise in 2nd row if jth element is 0 go to next row. So Finally say if you are at any ith row and jth column which is index of last 1 from right in that row, increment i. So now if we have Aij = 0 again increment i otherwise keep decreasing j until you find last 1 in that particular row.
Sample Illustration :

Algorithm :

```
for (int i=0, j=n-1; i<n;i++)
{
    // find left most position of 1 in a row
    // find 1st zero in a row
    while (arr[i][j]==1)
    {
        row = i;
        j--;
    }
}
cout << "Row number =" << row+1;
cout << "MaxCount =" << n-j;
```

C++

```
 // CPP program to find row with maximum 1
// in row sorted binary matrix
#include<bits/stdc++.h>
#define N 4
using namespace std;

// function for finding row with maximum 1
void findMax (int arr[][N])
{
    int row = 0, i, j;
    for (i=0, j=N-1; i<N;i++)
    {
        // find left most position of 1 in a row
        // find 1st zero in a row
        while (arr[i][j] == 1 && j >= 0)
        {
            row = i;
            j--;
        }
    }
```

```
    cout << "Row number = " << row+1;
    cout << ", MaxCount = " << N-1-j;
}

// driver program
int main()
{
    int arr[N][N] = {0, 0, 0, 1,
                     0, 0, 0, 1,
                     0, 0, 0, 0,
                     0, 1, 1, 1};
    findMax(arr);
    return 0;
}
```

Java

```
 // Java program to find row with maximum 1
// in row sorted binary matrix
class GFG {

    static final int N = 4;

    // function for finding row with maximum 1
    static void findMax(int arr[][]) {

        int row = 0, i, j;

        for (i = 0, j = N - 1; i < N; i++) {

            // find left most position of 1 in
            // a row find 1st zero in a row
            while (arr[i][j] == 1 && j >= 0) {

                row = i;
                j--;
            }
        }

        System.out.print("Row number = "
                                + (row + 1));
        System.out.print(", MaxCount = "
                               + (N - 1 - j));
    }

    // Driver code
    public static void main(String[] args) {
        int arr[][] = {{0, 0, 0, 1},
```

```
                        {0, 0, 0, 1},
                        {0, 0, 0, 0},
                        {0, 1, 1, 1}};
        findMax(arr);
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # python program to find row with
# maximum 1 in row sorted binary
# matrix

N = 4

# function for finding row with
# maximum 1
def findMax (arr):
    row = 0
    j = N - 1
    for i in range(0, N):
        # find left most position
        # of 1 in a row find 1st
        # zero in a row
        while (arr[i][j] == 1
                    and j >= 0):
            row = i
            j -= 1

    print("Row number = " , row + 1,
          ", MaxCount = ", N - 1 - j)

# driver program
arr = [ [0, 0, 0, 1],
        [0, 0, 0, 1],
        [0, 0, 0, 0],
        [0, 1, 1, 1] ]

findMax(arr)

# This code is contributed by Sam007
```

C#

```
 // C# program to find row with maximum
```

```
// 1 in row sorted binary matrix
using System;

class GFG {

    static int N = 4;

    // function for finding row with maximum 1
    static void findMax(int [,]arr)
    {
        int row = 0, i, j;

        for (i = 0, j = N - 1; i < N; i++) {

            // find left most position of 1 in
            // a row find 1st zero in a row
            while (arr[i,j] == 1 && j >= 0)
            {
                row = i;
                j--;
            }
        }

        Console.Write("Row number = " + (row + 1));
        Console.Write(", MaxCount = " + (N - 1 - j));
    }

    // Driver code
    public static void Main()
    {
        int [,]arr = {{0, 0, 0, 1},
                      {0, 0, 0, 1},
                      {0, 0, 0, 0},
                      {0, 1, 1, 1}};
        findMax(arr);
    }
}

// This code is contributed by nitin mittal
```

PHP

```
 <?php
// PHP program to find
// row with maximum 1
// in row sorted
// binary matrix
$N = 4;
```

```
// function for finding
// row with maximum 1
function findMax ($arr)
{

    global $N;
    $row = 0; $i;
    $j=$N - 1;
    for ($i = 0; $i < $N; $i++)
    {

        // find left most position
        // of 1 in a row find 1st
        // zero in a row
        while ($arr[$i][$j] == 1 &&
                           $j >= 0)
        {
            $row = $i;
            $j--;
        }
    }
    echo "Row number = " , $row + 1;
    echo ", MaxCount = " , $N - 1 - $j;
}

    // Driver Code
    $arr = array(array(0, 0, 0, 1),
                 array(0, 0, 0, 1),
                 array(0, 0, 0, 0),
                 array(0, 1, 1, 1));
    findMax($arr);

// This code is contributed by vt_m.
?>
```

Output:

```
Row number = 4, MaxCount = 3
```

Improved By : nitin mittal, vt_m, Sam007

Source

https://www.geeksforgeeks.org/find-row-number-binary-matrix-maximum-number-1s/

Chapter 83

Find the Missing Number

Find the Missing Number - GeeksforGeeks

You are given a list of n-1 integers and these integers are in the range of 1 to n. There are no duplicates in list. One of the integers is missing in the list. Write an efficient code to find the missing integer.

```
Example :
I/P    [1, 2, 4, ,6, 3, 7, 8]
O/P    5
```

METHOD 1(Use sum formula)
Algorithm:

```
1. Get the sum of numbers
       total = n*(n+1)/2
2  Subtract all the numbers from sum and
   you will get the missing number.
```

C

```
 #include<stdio.h>

/* getMissingNo takes array and size of array as arguments*/
int getMissingNo (int a[], int n)
{
    int i, total;
    total  = (n+1)*(n+2)/2;
    for ( i = 0; i< n; i++)
```

```
       total -= a[i];
    return total;
}

/*program to test above function */
int main()
{
    int a[] = {1,2,4,5,6};
    int miss = getMissingNo(a,5);
    printf("%d", miss);
    getchar();
}
```

Java

```
 // Java program to find missing Number

class Main
{
    // Function to ind missing number
    static int getMissingNo (int a[], int n)
    {
        int i, total;
        total  = (n+1)*(n+2)/2;
        for ( i = 0; i< n; i++)
           total -= a[i];
        return total;
    }

    /* program to test above function */
    public static void main(String args[])
    {
        int a[] = {1,2,4,5,6};
        int miss = getMissingNo(a,5);
        System.out.println(miss);
    }
}
```

Python

```
 # getMissingNo takes list as argument
def getMissingNo(A):
    n = len(A)
    total = (n+1)*(n+2)/2
    sum_of_A = sum(A)
    return total - sum_of_A
```

```
# Driver program to test above function
A = [1, 2, 4, 5, 6]
miss = getMissingNo(A)
print(miss)
# This code is contributed by Pratik Chhajer
```

C#

```
 // C# program to find missing Number
using System;

class GFG
{
    // Function to ind missing number
    static int getMissingNo (int []a, int n)
    {
        int total = (n + 1) * (n + 2) / 2;

        for (int i = 0; i< n; i++)
        total -= a[i];

        return total;
    }

    /* program to test above function */
    public static void Main()
    {
        int []a = {1, 2, 4, 5, 6};
        int miss = getMissingNo(a, 5);
        Console.Write(miss);
    }

}

// This code is contributed by Sam007_
```

PHP

```
 <?php
// PHP program to find
// the Missing Number

// getMissingNo takes array and
// size of array as arguments
function getMissingNo ($a, $n)
{
    $total = ($n + 1) * ($n + 2) / 2;
```

```
    for ( $i = 0; $i < $n; $i++)
        $total -= $a[$i];
    return $total;
}

// Driver Code
$a = array(1, 2, 4, 5, 6);
$miss = getMissingNo($a, 5);
echo($miss);

// This code is contributed by Ajit.
?>
```

Output :

```
3
```

Time Complexity : O(n)

There can be overflow if n is large. In order to avoid Integer Overflow, we can pick one number from known numbers and subtract one number from given numbers. This way we wont have Integer Overflow ever. Thanks to Sahil Rally for suggesting this improvement.

METHOD 2(Use XOR)

```
  1) XOR all the array elements, let the result of XOR be X1.
  2) XOR all numbers from 1 to n, let XOR be X2.
  3) XOR of X1 and X2 gives the missing number.
```

C

```
 #include<stdio.h>

/* getMissingNo takes array and size of array as arguments*/
int getMissingNo(int a[], int n)
{
    int i;
    int x1 = a[0]; /* For xor of all the elements in array */
    int x2 = 1; /* For xor of all the elements from 1 to n+1 */

    for (i = 1; i< n; i++)
        x1 = x1^a[i];

    for ( i = 2; i <= n+1; i++)
        x2 = x2^i;
```

```
    return (x1^x2);
}

/*program to test above function */
int main()
{
    int a[] = {1, 2, 4, 5, 6};
    int miss = getMissingNo(a, 5);
    printf("%d", miss);
    getchar();
}
```

Java

```
 // Java program to find missing Number
// using xor
class Main
{
    // Function to find missing number
    static int getMissingNo (int a[], int n)
    {
        int x1 = a[0];
        int x2 = 1;

        /* For xor of all the elements
           in array */
        for (int i = 1; i < n; i++)
            x1 = x1 ^ a[i];

        /* For xor of all the elements
           from 1 to n+1 */
        for (int i = 2; i <= n+1; i++)
            x2 = x2 ^ i;

        return (x1 ^ x2);
    }

    /* program to test above function */
    public static void main(String args[])
    {
        int a[] = {1,2,4,5,6};
        int miss = getMissingNo(a,5);
        System.out.println(miss);
    }
}
```

C#

```
 // C# program to find missing Number
// using xor
using System;

class GFG
{
    // Function to find missing number
    static int getMissingNo (int []a, int n)
    {
        int x1 = a[0];
        int x2 = 1;

        /* For xor of all the elements
        in array */
        for (int i = 1; i < n; i++)
            x1 = x1 ^ a[i];

        /* For xor of all the elements
        from 1 to n+1 */
        for (int i = 2; i <= n + 1; i++)
            x2 = x2 ^ i;

        return (x1 ^ x2);
    }

    /* driver program to test above function */
    public static void Main()
    {
        int []a = {1, 2, 4, 5, 6};
        int miss = getMissingNo(a, 5);
        Console.Write(miss);
    }

}

// This code is contributed by Sam007_
```

PHP

```
 <?php
// PHP program to find
// the Misiing Number
// getMissingNo takes array and
// size of array as arguments
function getMissingNo($a, $n)
{
    // For xor of all the
    // elements in array
```

```
    $x1 = $a[0];

    // For xor of all the
    // elements from 1 to n + 1
    $x2 = 1;

    for ($i = 1; $i < $n; $i++)
        $x1 = $x1 ^ $a[$i];

    for ($i = 2; $i <= $n + 1; $i++)
        $x2 = $x2 ^ $i;

    return ($x1 ^ $x2);
}

// Driver Code
$a = array(1, 2, 4, 5, 6);
$miss = getMissingNo($a, 5);
echo($miss);

// This code is contributed by Ajit.
?>
```

Output :

```
3
```

Time Complexity : O(n)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/find-the-missing-number/

Chapter 84

Find the Missing Number in a sorted array

Find the Missing Number in a sorted array - GeeksforGeeks

Given a list of n-1 integers and these integers are in the range of 1 to n. There are no duplicates in list. One of the integers is missing in the list. Write an efficient code to find the missing integer.

Examples:

```
Input : arr[] = [1, 2, 3, 4, 6, 7, 8]
Output : 5

Input : arr[] = [1, 2, 3, 4, 5, 6, 8, 9]
Output : 7
```

One **Simple solution** is to apply methods discussed for finding the missing element in an unsorted array. Time complexity of this solution is O(n).

An **efficient solution** is based on the divide and conquer algorithm that we have seen in binary search, the concept behind this solution is that the elements appearing before the missing element will have ar[i] – i = 1 and those appearing after the missing element will have ar[i] – i = 2.

This solution has a time complexity of O(log n)

C++

```
 // A binary search based program to find the
// only missing number in a sorted array of
// distinct elements within limited range.
```

```
#include <iostream>
using namespace std;

int search(int ar[], int size)
{
    int a = 0, b = size - 1;
    int mid;
    while ((b - a) > 1) {
        mid = (a + b) / 2;
        if ((ar[a] - a) != (ar[mid] - mid))
            b = mid;
        else if ((ar[b] - b) != (ar[mid] - mid))
            a = mid;
    }
    return (ar[mid] + 1);
}

int main()
{
    int ar[] = { 1, 2, 3, 4, 5, 6, 8 };
    int size = sizeof(ar) / sizeof(ar[0]);
    cout << "Missing number:" << search(ar, size);
}
```

Java

```
 // A binary search based program
// to find the only missing number
// in a sorted array of distinct
// elements within limited range.
import java.io.*;

class GFG
{
static int search(int ar[],
                  int size)
{
    int a = 0, b = size - 1;
    int mid = 0;
    while ((b - a) > 1)
    {
        mid = (a + b) / 2;
        if ((ar[a] - a) != (ar[mid] - mid))
            b = mid;
        else if ((ar[b] - b) != (ar[mid] - mid))
            a = mid;
    }
    return (ar[mid] + 1);
```

```
}

// Driver Code
public static void main (String[] args)
{
    int ar[] = { 1, 2, 3, 4, 5, 6, 8 };
    int size = ar.length;
    System.out.println("Missing number: " +
                        search(ar, size));
}
}

// This code is contributed
// by inder_verma.
```

C#

```
 // A binary search based program
// to find the only missing number
// in a sorted array of distinct
// elements within limited range.
using System;

class GFG
{
static int search(int []ar,
                  int size)
{
    int a = 0, b = size - 1;
    int mid = 0;
    while ((b - a) > 1)
    {
        mid = (a + b) / 2;
        if ((ar[a] - a) != (ar[mid] - mid))
            b = mid;
        else if ((ar[b] - b) != (ar[mid] - mid))
            a = mid;
    }
    return (ar[mid] + 1);
}

// Driver Code
static public void Main (String []args)
{
    int []ar = { 1, 2, 3, 4, 5, 6, 8 };
    int size = ar.Length;
    Console.WriteLine("Missing number: " +
                        search(ar, size));
```

```
}
}

// This code is contributed
// by Arnab Kundu
```

Output:

```
Missing number: 7
```

Improved By : inderDuMCA, andrew1234

Source

https://www.geeksforgeeks.org/find-the-missing-number-in-a-sorted-array/

Chapter 85

Find the closest pair from two sorted arrays

Find the closest pair from two sorted arrays - GeeksforGeeks

Given two sorted arrays and a number x, find the pair whose sum is closest to x and **the pair has an element from each array**.

We are given two arrays ar1[0…m-1] and ar2[0..n-1] and a number x, we need to find the pair ar1[i] + ar2[j] such that absolute value of (ar1[i] + ar2[j] – x) is minimum.

Example:

```
Input:  ar1[] = {1, 4, 5, 7};
        ar2[] = {10, 20, 30, 40};
        x = 32
Output:  1 and 30

Input:  ar1[] = {1, 4, 5, 7};
        ar2[] = {10, 20, 30, 40};
        x = 50
Output:  7 and 40
```

We strongly recommend to minimize your browser and try this yourself first.

A **Simple Solution** is to run two loops. The outer loop considers every element of first array and inner loop checks for the pair in second array. We keep track of minimum difference between ar1[i] + ar2[j] and x.

We can do it **in O(n) time** using following steps.
1) Merge given two arrays into an auxiliary array of size m+n using merge process of merge sort. While merging keep another boolean array of size m+n to indicate whether the current element in merged array is from ar1[] or ar2[].

2) Consider the merged array and use the linear time algorithm to find the pair with sum closest to x. One extra thing we need to consider only those pairs which have one element from ar1[] and other from ar2[], we use the boolean array for this purpose.

Can we do it in a single pass and O(1) extra space?
The idea is to start from left side of one array and right side of another array, and use the algorithm same as step 2 of above approach. Following is detailed algorithm.

```
1) Initialize a variable diff as infinite (Diff is used to store the
   difference between pair and x).  We need to find the minimum diff.
2) Initialize two index variables l and r in the given sorted array.
       (a) Initialize first to the leftmost index in ar1:  l = 0
       (b) Initialize second  the rightmost index in ar2:  r = n-1
3) Loop while  l = 0
       (a) If  abs(ar1[l] + ar2[r] - sum) < diff  then
           update diff and result
       (b) Else if(ar1[l] + ar2[r] <  sum )  then l++
       (c) Else r--
4) Print the result.
```

Following is the implementation of this approach.

C++

```
 // C++ program to find the pair from two sorted arays such
// that the sum of pair is closest to a given number x
#include <iostream>
#include <climits>
#include <cstdlib>
using namespace std;

// ar1[0..m-1] and ar2[0..n-1] are two given sorted arrays
// and x is given number. This function prints the pair  from
// both arrays such that the sum of the pair is closest to x.
void printClosest(int ar1[], int ar2[], int m, int n, int x)
{
    // Initialize the diff between pair sum and x.
    int diff = INT_MAX;

    // res_l and res_r are result indexes from ar1[] and ar2[]
    // respectively
    int res_l, res_r;

    // Start from left side of ar1[] and right side of ar2[]
    int l = 0, r = n-1;
    while (l<m && r>=0)
```

```
    {
       // If this pair is closer to x than the previously
       // found closest, then update res_l, res_r and diff
       if (abs(ar1[l] + ar2[r] - x) < diff)
       {
           res_l = l;
           res_r = r;
           diff = abs(ar1[l] + ar2[r] - x);
       }

       // If sum of this pair is more than x, move to smaller
       // side
       if (ar1[l] + ar2[r] > x)
           r--;
       else  // move to the greater side
           l++;
    }

    // Print the result
    cout << "The closest pair is [" << ar1[res_l] << ", "
         << ar2[res_r] << "] \n";
}

// Driver program to test above functions
int main()
{
    int ar1[] = {1, 4, 5, 7};
    int ar2[] = {10, 20, 30, 40};
    int m = sizeof(ar1)/sizeof(ar1[0]);
    int n = sizeof(ar2)/sizeof(ar2[0]);
    int x = 38;
    printClosest(ar1, ar2, m, n, x);
    return 0;
}
```

Java

```
 // Java program to find closest pair in an array
class ClosestPair
{
    // ar1[0..m-1] and ar2[0..n-1] are two given sorted
    // arrays/ and x is given number. This function prints
    // the pair from both arrays such that the sum of the
    // pair is closest to x.
    void printClosest(int ar1[], int ar2[], int m, int n, int x)
    {
        // Initialize the diff between pair sum and x.
        int diff = Integer.MAX_VALUE;
```

```
        // res_l and res_r are result indexes from ar1[] and ar2[]
        // respectively
        int res_l = 0, res_r = 0;

        // Start from left side of ar1[] and right side of ar2[]
        int l = 0, r = n-1;
        while (l<m && r>=0)
        {
           // If this pair is closer to x than the previously
           // found closest, then update res_l, res_r and diff
           if (Math.abs(ar1[l] + ar2[r] - x) < diff)
           {
               res_l = l;
               res_r = r;
               diff = Math.abs(ar1[l] + ar2[r] - x);
           }

           // If sum of this pair is more than x, move to smaller
           // side
           if (ar1[l] + ar2[r] > x)
               r--;
           else  // move to the greater side
               l++;
        }

        // Print the result
        System.out.print("The closest pair is [" + ar1[res_l] +
                         ", " + ar2[res_r] + "]");
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
        ClosestPair ob = new ClosestPair();
        int ar1[] = {1, 4, 5, 7};
        int ar2[] = {10, 20, 30, 40};
        int m = ar1.length;
        int n = ar2.length;
        int x = 38;
        ob.printClosest(ar1, ar2, m, n, x);
    }
}
/*This code is contributed by Rajat Mishra */
```

Python3

```
 # Python3 program to find the pair from
```

```
# two sorted arays such that the sum
# of pair is closest to a given number x
import sys

# ar1[0..m-1] and ar2[0..n-1] are two
# given sorted arrays and x is given
# number. This function prints the pair
# from both arrays such that the sum
# of the pair is closest to x.
def printClosest(ar1, ar2, m, n, x):

    # Initialize the diff between
    # pair sum and x.
    diff=sys.maxsize

    # res_l and res_r are result
    # indexes from ar1[] and ar2[]
    # respectively. Start from left
    # side of ar1[] and right side of ar2[]
    l = 0
    r = n-1
    while(l < m and r >= 0):

    # If this pair is closer to x than
    # the previously found closest,
    # then update res_l, res_r and diff
        if abs(ar1[l] + ar2[r] - x) < diff:
            res_l = l
            res_r = r
            diff = abs(ar1[l] + ar2[r] - x)


    # If sum of this pair is more than x,
    # move to smaller side
        if ar1[l] + ar2[r] > x:
            r=r-1
        else: # move to the greater side
            l=l+1


    # Print the result
    print("The closest pair is [",
         ar1[res_l],",",ar2[res_r],"]")

# Driver program to test above functions
ar1 = [1, 4, 5, 7]
ar2 = [10, 20, 30, 40]
m = len(ar1)
```

```
n = len(ar2)
x = 38
printClosest(ar1, ar2, m, n, x)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to find closest pair in
// an array
using System;

class GFG {

    // ar1[0..m-1] and ar2[0..n-1] are two
    // given sorted arrays/ and x is given
    // number. This function prints the
    // pair from both arrays such that the
    // sum of the pair is closest to x.
    static void printClosest(int []ar1,
            int []ar2, int m, int n, int x)
    {

        // Initialize the diff between pair
        // sum and x.
        int diff = int.MaxValue;

        // res_l and res_r are result
        // indexes from ar1[] and ar2[]
        // respectively
        int res_l = 0, res_r = 0;

        // Start from left side of ar1[]
        // and right side of ar2[]
        int l = 0, r = n-1;
        while (l < m && r >= 0)
        {

            // If this pair is closer to
            // x than the previously
            // found closest, then update
            // res_l, res_r and diff
            if (Math.Abs(ar1[l] +
                       ar2[r] - x) < diff)
            {
                res_l = l;
                res_r = r;
                diff = Math.Abs(ar1[l]
```

```
                            + ar2[r] - x);
            }

            // If sum of this pair is more
            // than x, move to smaller
            // side
            if (ar1[l] + ar2[r] > x)
                r--;
            else // move to the greater side
                l++;
        }

        // Print the result
        Console.Write("The closest pair is ["
                        + ar1[res_l] + ", "
                       + ar2[res_r] + "]");
    }

    // Driver program to test above functions
    public static void Main()
    {
        int []ar1 = {1, 4, 5, 7};
        int []ar2 = {10, 20, 30, 40};
        int m = ar1.Length;
        int n = ar2.Length;
        int x = 38;

        printClosest(ar1, ar2, m, n, x);
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to find the pair
// from two sorted arays such
// that the sum of pair is
// closest to a given number x

// ar1[0..m-1] and ar2[0..n-1]
// are two given sorted arrays
// and x is given number. This
// function prints the pair from
// both arrays such that the sum
// of the pair is closest to x.
function printClosest($ar1, $ar2,
```

```
                    $m, $n, $x)
{

    // Initialize the diff between
    // pair sum and x.
    $diff = PHP_INT_MAX;

    // res_l and res_r are result
    // indexes from ar1[] and ar2[]
    // respectively
    $res_l;
    $res_r;

    // Start from left side of
    // ar1[] and right side of ar2[]
    $l = 0;
    $r = $n - 1;
    while ($l < $m and $r >= 0)
    {

        // If this pair is closer to
        // x than the previously
        // found closest, then
        // update res_l, res_r and
        // diff
        if (abs($ar1[$l] + $ar2[$r] -
                        $x) < $diff)
        {
            $res_l = $l;
            $res_r = $r;
            $diff = abs($ar1[$l] +
                    $ar2[$r] - $x);
        }

        // If sum of this pair is
        // more than x, move to smaller
        // side
        if ($ar1[$l] + $ar2[$r] > $x)
            $r--;

        // move to the greater side
        else
            $l++;
    }

    // Print the result
    echo "The closest pair is [" , $ar1[$res_l] , ", "
                               , $ar2[$res_r] , "] \n";
```

```
}

    // Driver Code
    $ar1 = array(1, 4, 5, 7);
    $ar2 = array(10, 20, 30, 40);
    $m = count($ar1);
    $n = count($ar2);
    $x = 38;
    printClosest($ar1, $ar2, $m, $n, $x);

// This code is contributed by anuj_67.
?>
```

Output:

```
 The closest pair is [7, 30]
```

Smallest Difference pair of values between two unsorted Arrays

This article is contributed by Harsh. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/given-two-sorted-arrays-number-x-find-pair-whose-sum-closest-x/

Chapter 86

Find the element before which all the elements are smaller than it, and after which all are greater

Find the element before which all the elements are smaller than it, and after which all are greater - GeeksforGeeks

Given an array, find an element before which all elements are smaller than it, and after which all are greater than it. Return index of the element if there is such an element, otherwise return -1.

Examples:

```
Input:   arr[] = {5, 1, 4, 3, 6, 8, 10, 7, 9};
Output:  Index of element is 4
All elements on left of arr[4] are smaller than it
and all elements on right are greater.

Input:   arr[] = {5, 1, 4, 4};
Output:  Index of element is -1
```

Expected time complexity is O(n).

A **Simple Solution** is to consider every element one by one. For every element, compare it with all elements on left and all elements on right. Time complexity of this solution is $O(n^2)$.

An **Efficient Solution** can solve this problem in O(n) time using O(n) extra space. Below is detailed solution.

1) Create two arrays leftMax[] and rightMin[].
2) Traverse input array from left to right and fill leftMax[] such that leftMax[i] contains maximum element from 0 to i-1 in input array.
3) Traverse input array from right to left and fill rightMin[] such that rightMin[i] contains minimum element from to n-1 to i+1 in input array.
4) Traverse input array. For every element arr[i], check if arr[i] is greater than leftMax[i] and smaller than rightMin[i]. If yes, return i.

Further Optimization to above approach is to use only one extra array and traverse input array only twice. First traversal is same as above and fills leftMax[]. Next traversal traverses from right and keeps track of minimum. The second traversal also finds the required element.

Below is C++ implementation of above approach.

```
 // C++ program to find the element which is greater than
// all left elements and smaller than all right elements.
#include <bits/stdc++.h>
using namespace std;

int findElement(int arr[], int n)
{
    // leftMax[i] stores maximum of arr[0..i-1]
    int leftMax[n];
    leftMax[0] = INT_MIN;

    // Fill leftMax[]1..n-1]
    for (int i = 1; i < n; i++)
        leftMax[i] = max(leftMax[i-1], arr[i-1]);

    // Initialize minimum from right
    int rightMin = INT_MAX;

    // Traverse array from right
    for (int i=n-1; i>=0; i--)
    {
        // Check if we found a required element
        if (leftMax[i] < arr[i] && rightMin > arr[i])
             return i;

        // Update right minimum
        rightMin = min(rightMin, arr[i]);
    }

    // If there was no element matching criteria
    return -1;
}

// Driver program
int main()
```

```
{
    int arr[] = {5, 1, 4, 3, 6, 8, 10, 7, 9};
    int n = sizeof arr / sizeof arr[0];
    cout << "Index of the element is " << findElement(arr, n);
    return 0;
}
```

Index of the element is 4.

Time Complexity: O(n)
Auxiliary Space: O(n)

Thanks to Gaurav Ahirwar for suggesting above solution.

Source

https://www.geeksforgeeks.org/find-the-element-before-which-all-the-elements-are-smaller-than-it-and-after-which

Chapter 87

Find the element that appears once in a sorted array

Find the element that appears once in a sorted array - GeeksforGeeks

Given a sorted array in which all elements appear twice (one after one) and one element appears only once. Find that element in O(log n) complexity.

Example:

```
Input:   arr[] = {1, 1, 3, 3, 4, 5, 5, 7, 7, 8, 8}
Output:  4

Input:   arr[] = {1, 1, 3, 3, 4, 4, 5, 5, 7, 7, 8}
Output:  8
```

A **Simple Solution** is to traverse the array from left to right. Since the array is sorted, we can easily figure out the required element.

An **Efficient Solution** can find the required element in O(Log n) time. The idea is to use Binary Search. Below is an observation in input array.
All elements before the required have first occurrence at even index (0, 2, ..) and next occurrence at odd index (1, 3, …). And all elements after the required element have first occurrence at odd index and next occurrence at even index.

1) Find the middle index, say 'mid'.

2) If 'mid' is even, then compare arr[mid] and arr[mid + 1]. If both are same, then the required element after 'mid' else before mid.

3) If 'mid' is odd, then compare arr[mid] and arr[mid – 1]. If both are same, then the required element after 'mid' else before mid.

Below is the implementation based on above idea.

C/C++

```
 // C program to find the element that appears only once
#include<stdio.h>

// A Binary Search based function to find the element
// that appears only once
void search(int *arr, int low, int high)
{
     // Base cases
    if (low > high)
       return;

    if (low==high)
    {
        printf("The required element is %d ", arr[low]);
        return;
    }

    // Find the middle point
    int mid = (low + high) / 2;

    // If mid is even and element next to mid is
    // same as mid, then output element lies on
    // right side, else on left side
    if (mid%2 == 0)
    {
        if (arr[mid] == arr[mid+1])
            search(arr, mid+2, high);
        else
            search(arr, low, mid);
    }
    else  // If mid is odd
    {
        if (arr[mid] == arr[mid-1])
            search(arr, mid+1, high);
        else
            search(arr, low, mid-1);
    }
}

// Driver program
int main()
{
    int arr[] = {1, 1, 2, 4, 4, 5, 5, 6, 6};
    int len = sizeof(arr)/sizeof(arr[0]);
    search(arr, 0, len-1);
    return 0;
```

```
}
```

Java

```
 // Java program to find the element that appears only once

public class Main
{
    // A Binary Search based method to find the element
    // that appears only once
    public static void search(int[] arr, int low, int high)
    {
        if(low > high)
            return;
        if(low == high)
        {
            System.out.println("The required element is "+arr[low]);
            return;
        }

        // Find the middle point
        int mid = (low + high)/2;

        // If mid is even and element next to mid is
        // same as mid, then output element lies on
        // right side, else on left side
        if(mid % 2 == 0)
        {
            if(arr[mid] == arr[mid+1])
                search(arr, mid+2, high);
            else
                search(arr, low, mid);
        }
        // If mid is odd
        else if(mid % 2 == 1)
        {
            if(arr[mid] == arr[mid-1])
                search(arr, mid+1, high);
            else
                search(arr, low, mid-1);
        }
    }

    public static void main(String[] args)
    {
        int[] arr = {1, 1, 2, 4, 4, 5, 5, 6, 6};
        search(arr, 0, arr.length-1);
    }
```

```
}
// This code is contributed by Tanisha Mittal
```

Python

```
 # A Binary search based function to find
# the element that appears only once
def search(arr, low, high):

    # Base cases
    if low > high:
        return None

    if low == high:
        return arr[low]

    # Find the middle point
    mid = low + (high - low)/2

    # If mid is even and element next to mid is
    # same as mid, then output element lies on
    # right side, else on left side
    if mid%2 == 0:

        if arr[mid] == arr[mid+1]:
            return search(arr, mid+2, high)
        else:
            return search(arr, low, mid)

    else:
        # if mid is odd
        if arr[mid] == arr[mid-1]:
            return search(arr, mid+1, high)
        else:
            return search(arr, low, mid-1)


# Test Array
arr = [ 1, 1, 2, 4, 4, 5, 5, 6, 6 ]

# Function call
result = search(arr, 0, len(arr)-1)

if result is not None:
    print "The required element is %d" % result
else:
    print "Invalid Array"
```

C#

```
 // C# program to find the element
// that appears only once
using System;

class GFG {

    // A Binary Search based
    // method to find the element
    // that appears only once
    public static void search(int[] arr,
                              int low,
                              int high)
    {

        if(low > high)
            return;
        if(low == high)
        {
            Console.WriteLine("The required element is "
                                            +arr[low]);
            return;
        }

        // Find the middle point
        int mid = (low + high)/2;

        // If mid is even and element
        // next to mid is same as mid
        // then output element lies on
        // right side, else on left side
        if(mid % 2 == 0)
        {
            if(arr[mid] == arr[mid + 1])
                search(arr, mid + 2, high);
            else
                search(arr, low, mid);
        }

        // If mid is odd
        else if(mid % 2 == 1)
        {
            if(arr[mid] == arr[mid - 1])
                search(arr, mid + 1, high);
            else
                search(arr, low, mid - 1);
        }
```

```
    }

    // Driver Code
    public static void Main(String[] args)
    {
        int[] arr = {1, 1, 2, 4, 4, 5, 5, 6, 6};
        search(arr, 0, arr.Length - 1);
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP program to find the element
// that appears only once

// A Binary Search based function
// to find the element that
// appears only once
function search($arr, $low, $high)
{

    // Base cases
    if ($low > $high)
        return;

    if ($low==$high)
    {
        echo("The required element is " );
        echo $arr[$low] ;
        return;
    }

    // Find the middle point
    $mid = ($low + $high) / 2;

    // If mid is even and element
    // next to mid is same as mid,
    // then output element lies on
    // right side, else on left side
    if ($mid % 2 == 0)
    {
        if ($arr[$mid] == $arr[$mid + 1])
            search($arr, $mid + 2, $high);
        else
            search($arr, $low, $mid);
```

```
    }

    // If mid is odd
    else
    {
        if ($arr[$mid] == $arr[$mid - 1])
            search($arr, $mid + 1, $high);
        else
            search($arr, $low, $mid - 1);
    }
}

    // Driver Code
    $arr = array(1, 1, 2, 4, 4, 5, 5, 6, 6);
    $len = sizeof($arr);
    search($arr, 0, $len - 1);

// This code is contributed by nitin mittal
?>
```

Output:

```
The required element is 2
```

Time Complexity: O(Log n)

This article is contributed by Mehboob Elahi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : Aalaa Abdel Mawla, nitin mittal, AntonyCherepanov

Source

https://www.geeksforgeeks.org/find-the-element-that-appears-once-in-a-sorted-array/

Chapter 88

Find the first repeating element in an array of integers

Find the first repeating element in an array of integers - GeeksforGeeks

Given an array of integers, find the first repeating element in it. We need to find the element that occurs more than once and whose index of first occurrence is smallest.

Examples:

```
Input:  arr[] = {10, 5, 3, 4, 3, 5, 6}
Output: 5 [5 is the first element that repeats]

Input:  arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10}
Output: 6 [6 is the first element that repeats]
```

A **Simple Solution** is to use two nested loops. The outer loop picks an element one by one, the inner loop checks whether the element is repeated or not. Once we find an element that repeats, we break the loops and print the element. Time Complexity of this solution is $O(n^2)$

We can **Use Sorting** to solve the problem in O(nLogn) time. Following are detailed steps.
1) Copy the given array to an auxiliary array temp[].
2) Sort the temp array using a O(nLogn) time sorting algorithm.
3) Scan the input array from left to right. For every element, count its occurrences in temp[] using binary search. As soon as we find an element that occurs more than once, we return the element. This step can be done in O(nLogn) time.

We can **Use Hashing** to solve this in O(n) time on average. The idea is to traverse the given array from right to left and update the minimum index whenever we find an element that has been visited on right side. Thanks to Mohammad Shahid for suggesting this solution.

Following are C++ and Java implementation of this idea.

C++

```
 /* C++ program to find first repeating element in arr[] */
#include<bits/stdc++.h>
using namespace std;

// This function prints the first repeating element in arr[]
void printFirstRepeating(int arr[], int n)
{
    // Initialize index of first repeating element
    int min = -1;

    // Creates an empty hashset
    set<int> myset;

    // Traverse the input array from right to left
    for (int i = n - 1; i >= 0; i--)
    {
        // If element is already in hash set, update min
        if (myset.find(arr[i]) != myset.end())
            min = i;

        else   // Else add element to hash set
            myset.insert(arr[i]);
    }

    // Print the result
    if (min != -1)
        cout << "The first repeating element is " << arr[min];
    else
        cout << "There are no repeating elements";
}

// Driver method to test above method
int main()
{
    int arr[] = {10, 5, 3, 4, 3, 5, 6};

    int n = sizeof(arr) / sizeof(arr[0]);
    printFirstRepeating(arr, n);
}
//This article is contributed by Chhavi
```

Java

```
 /* Java program to find first repeating element in arr[] */
import java.util.*;

class Main
{
```

```
    // This function prints the first repeating element in arr[]
    static void printFirstRepeating(int arr[])
    {
        // Initialize index of first repeating element
        int min = -1;

        // Creates an empty hashset
        HashSet<Integer> set = new HashSet<>();

        // Traverse the input array from right to left
        for (int i=arr.length-1; i>=0; i--)
        {
            // If element is already in hash set, update min
            if (set.contains(arr[i]))
                min = i;

            else   // Else add element to hash set
                set.add(arr[i]);
        }

        // Print the result
        if (min != -1)
          System.out.println("The first repeating element is " + arr[min]);
        else
          System.out.println("There are no repeating elements");
    }

    // Driver method to test above method
    public static void main (String[] args) throws java.lang.Exception
    {
        int arr[] = {10, 5, 3, 4, 3, 5, 6};
        printFirstRepeating(arr);
    }
}
```

Output:

```
The first repeating element is 5
```

Source

https://www.geeksforgeeks.org/find-first-repeating-element-array-integers/

Chapter 89

Find the first, second and third minimum elements in an array

Find the first, second and third minimum elements in an array - GeeksforGeeks

Find the first, second and third minimum elements in an array in O(n).

Examples:

```
Input : 9 4 12 6
Output : First min = 4
         Second min = 6
         Third min = 9

Input : 4 9 1 32 12
Output : First min = 1
         Second min = 4
         Third min = 9
```

First approach : First we can use normal method that is sort the array and then print first, second and third element of the array. Time complexity of this solution is O(n Log n).

Second approach : Time complexity of this solution is O(n).

Algorithm-

```
First take an element
then if array[index] < Firstelement
        Thirdelement = Secondelement
        Secondelement = Firstelement
        Firstelement = array[index]
     else if array[index] < Secondelement
```

```
        Thirdelement = Secondelement
        Secondelement = array[index]
     else if array[index] < Thirdelement
        Thirdelement = array[index]

then print all the element
```

C++

```
 // CPP program to find the first, second
// and third minimum element in an array
#include<iostream>
#define MAX 100000
using namespace std;

int Print3Smallest(int array[], int n)
{
    int firstmin = MAX, secmin = MAX, thirdmin = MAX;
    for (int i = 0; i < n; i++)
    {
        /* Check if current element is less than
           firstmin, then update first, second and
           third */
        if (array[i] < firstmin)
        {
            thirdmin = secmin;
            secmin = firstmin;
            firstmin = array[i];
        }

        /* Check if current element is less than
        secmin then update second and third */
        else if (array[i] < secmin)
        {
            thirdmin = secmin;
            secmin = array[i];
        }

        /* Check if current element is less than
        then update third */
        else if (array[i] < thirdmin)
            thirdmin = array[i];
    }

    cout << "First min = " << firstmin << "\n";
    cout << "Second min = " << secmin << "\n";
    cout << "Third min = " << thirdmin << "\n";
}
```

```
// Driver code
int main()
{
    int array[] = {4, 9, 1, 32, 12};
    int n = sizeof(array) / sizeof(array[0]);
    Print3Smallest(array, n);
    return 0;
}
```

Java

```
 // Java program to find the first, second
// and third minimum element in an array
import java.util.*;

public class GFG
{
    static void Print3Smallest(int array[], int n)
    {
            int firstmin = Integer.MAX_VALUE;
            int secmin = Integer.MAX_VALUE;
            int thirdmin = Integer.MAX_VALUE;
            for (int i = 0; i < n; i++)
            {
                /* Check if current element is less than
                firstmin, then update first, second and
                third */
                if (array[i] < firstmin)
                {
                    thirdmin = secmin;
                    secmin = firstmin;
                    firstmin = array[i];
                }

                /* Check if current element is less than
                secmin then update second and third */
                else if (array[i] < secmin)
                {
                    thirdmin = secmin;
                    secmin = array[i];
                }

                /* Check if current element is less than
                then update third */
                else if (array[i] < thirdmin)
                    thirdmin = array[i];
            }
```

```
            System.out.println("First min = " + firstmin );
            System.out.println("Second min = " + secmin );
            System.out.println("Third min = " + thirdmin );
    }

    // Driver code
    public static void main(String[] args)
    {
            int array[] = {4, 9, 1, 32, 12};
            int n = array.length;
            Print3Smallest(array, n);
    }

}

// This code is contributed by
// Sam007
```

Python3

```
 # A Python program to find the first,
# second and third minimum element
# in an array

MAX = 100000

def Print3Smallest(arr, n):
    firstmin = MAX
    secmin = MAX
    thirdmin = MAX

    for i in range(0, n):

        # Check if current element
        # is less than firstmin,
        # then update first,second
        # and third

        if arr[i] < firstmin:
            thirdmin = secmin
            secmin = firstmin
            firstmin = arr[i]

        # Check if current element is
        # less than secmin then update
        # second and third
        elif arr[i] < secmin:
```

```
            thirdmin = secmin
            secmin = arr[i]

        # Check if current element is
        # less than,then upadte third
        elif arr[i] < thirdmin:
            thirdmin = arr[i]

    print("First min = ", firstmin)
    print("Second min = ", secmin)
    print("Third min = ", thirdmin)


# driver program
arr = [4, 9, 1, 32, 12]
n = len(arr)
Print3Smallest(arr, n)

# This code is contributed by Shrikant13.
```

C#

```
 // C# program to find the first, second
// and third minimum element in an array
using System;

class GFG
{
static void Print3Smallest(int []array, int n)
    {
            int firstmin = int.MaxValue;
            int secmin = int.MaxValue;
            int thirdmin = int.MaxValue;

            for (int i = 0; i < n; i++)
            {
                /* Check if current element is less than
                firstmin, then update first, second and
                third */
                if (array[i] < firstmin)
                {
                    thirdmin = secmin;
                    secmin = firstmin;
                    firstmin = array[i];
                }

                /* Check if current element is less than
                secmin then update second and third */
```

```
                else if (array[i] < secmin)
                {
                    thirdmin = secmin;
                    secmin = array[i];
                }

                /* Check if current element is less than
                then update third */
                else if (array[i] < thirdmin)
                    thirdmin = array[i];
            }

            Console.WriteLine("First min = " + firstmin );
            Console.WriteLine("Second min = " + secmin );
            Console.WriteLine("Third min = " + thirdmin );
    }

    // Driver code
    static void Main()
    {
    int []array = new int[]{4, 9, 1, 32, 12};
    int n = array.Length;
    Print3Smallest(array, n);

    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// php program to find the first, second
// and third minimum element in an array

function Print3Smallest($array, $n)
{
    $MAX = 100000;
    $firstmin = $MAX;
    $secmin = $MAX;
    $thirdmin = $MAX;
    for ($i = 0; $i < $n; $i++)
    {

        /* Check if current element is less than
        firstmin, then update first, second and
        third */
        if ($array[$i] < $firstmin)
```

```
        {
            $thirdmin = $secmin;
            $secmin = $firstmin;
            $firstmin = $array[$i];
        }

        /* Check if current element is less than
        secmin then update second and third */
        else if ($array[$i] < $secmin)
        {
            $thirdmin = $secmin;
            $secmin = $array[$i];
        }

        /* Check if current element is less than
        then update third */
        else if ($array[$i] < $thirdmin)
            $thirdmin = $array[$i];
    }

    echo "First min = ".$firstmin."\n";
    echo "Second min = ".$secmin."\n";
    echo "Third min = ".$thirdmin."\n";
}

// Driver code
    $array= array(4, 9, 1, 32, 12);
    $n = sizeof($array) / sizeof($array[0]);
    Print3Smallest($array, $n);

// This code is contributed by mits
?>
```

Output:

```
First min = 1
Second min = 4
Third min = 9
```

Improved By : shrikanth13, Mithun Kumar

Source

https://www.geeksforgeeks.org/find-the-first-second-and-third-minimum-elements-in-an-array/

Chapter 90

Find the index of an array element in Java

Find the index of an array element in Java - GeeksforGeeks

Given an array of N elements and an element K, find the index of an array element in Java.

Examples:

```
Input: a[] = { 5, 4, 6, 1, 3, 2, 7, 8, 9 }, K = 5
Output: 0

Input: a[] = { 5, 4, 6, 1, 3, 2, 7, 8, 9 }, K = 7
Output: 6
```

An element in an array of N integers can be searched using the below-mentioned methods.

1. **Linear Search:** Doing a linear search in an array, the element can be found in O(N) complexity.

   Below is the implementation of the linear-search approach:

```
 // Java program to find index of
// an element in N elements
import java.util.*;
public class index {

    // Linear-search function to find the index of an element
    public static int findIndex(int arr[], int t)
    {

        // if array is Null
```

```
        if (arr == null) {
            return -1;
        }

        // find length of array
        int len = arr.length;
        int i = 0;

        // traverse in the array
        while (i < len) {

            // if the i-th element is t
            // then return the index
            if (arr[i] == t) {
                return i;
            }
            else {
                i = i + 1;
            }
        }
        return -1;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int[] my_array = { 5, 4, 6, 1, 3, 2, 7, 8, 9 };

        // find the index of 5
        System.out.println("Index position of 5 is: "
                           + findIndex(my_array, 5));

        // find the index of 7
        System.out.println("Index position of 7 is: "
                           + findIndex(my_array, 7));
    }
}
```

Output:

```
Index position of 5 is: 0
Index position of 7 is: 6
```

2. **Binary search:** Binary search can also be used to find the index of the array element in an array. But the binary search can only be used if the array is ***sorted***. Java

provides us with an inbuilt function which can be found in the Arrays library of Java which will rreturn the index if the element is present, else it returns -1. The complexity will be O(log n).

Below is the implementation of Binary search.

```
 // Java program to find index of
// an element in N elements
import java.util.Arrays;

public class index {

    // Function to find the index of an element
    public static int findIndex(int arr[], int t)
    {

        int index = Arrays.binarySearch(arr, t);
        return (index < 0) ? -1 : index;
    }
    // Driver Code
    public static void main(String[] args)
    {
        int[] my_array = { 1, 2, 3, 4, 5, 6, 7 };

        // find the index of 5
        System.out.println("Index position of 5 is: "
                           + findIndex(my_array, 5));

        // find the index of 7
        System.out.println("Index position of 7 is: "
                           + findIndex(my_array, 7));
    }
}
```

Output:

```
Index position of 5 is: 4
Index position of 7 is: 6
```

3. **Guava:** Guava is an open source, Java-based library developed by Google. It provides utility methods for collections, caching, primitives support, concurrency, common annotations, string processing, I/O, and validations. Guava provides several-utility class pertaining to be primitive like Ints for int, Longs for long, Doubles for double etc. Each utility class has an **indexOf()** method that returns the index of the first appearance of the element in array.

   Below is the implementation of Guava.

```
 // Java program to find index of
// an element in N elements
import java.util.List;
import com.google.common.primitives.Ints;
public class index {
    // Function to find the index of an element using
    public static int findIndex(int arr[], int t)
    {
        return Ints.indexOf(arr, t);
    }
    // Driver Code
    public static void main(String[] args)
    {
        int[] my_array = { 5, 4, 6, 1, 3, 2, 7, 8, 9 };
        System.out.println("Index position of 5 is: "
                           + findIndex(my_array, 5));
        System.out.println("Index position of 7 is: "
                           + findIndex(my_array, 7));
    }
}
```

Output:

```
Index position of 5 is: 0
Index position of 7 is: 6
```

4. **Stream API:** Stream is a new abstract layer introduced in Java 8. Using stream, you can process data in a declarative way similar to SQL statements. The stream represents a sequence of objects from a source, which supports aggregate operations. In order to find the index of an element Stream package provides utility, IntStream. Using the length of an array we can get an IntStream of array indices from 0 to n-1, where n is the length of an array.

   Below is the implementation of Stream API approach.

```
 // Java program to find index of
// an element in N elements
import java.util.stream.IntStream;
public class index {

    // Function to find the index of an element
    public static int findIndex(int arr[], int t)
    {
        int len = arr.length;
        return IntStream.range(0, len)
            .filter(i -> t == arr[i])
            .findFirst() // first occurence
```

```
                .orElse(-1); // No element found
    }
    public static void main(String[] args)
    {
        int[] my_array = { 5, 4, 6, 1, 3, 2, 7, 8, 9 };
        System.out.println("Index position of 5 is: "
                           + findIndex(my_array, 5));
        System.out.println("Index position of 7 is: "
                           + findIndex(my_array, 7));
    }
}
```

Output:

```
Index position of 5 is: 0
Index position of 7 is: 6
```

Source

https://www.geeksforgeeks.org/find-the-index-of-an-array-element-in-java/

Chapter 91

Find the k most frequent words from a file

Find the k most frequent words from a file - GeeksforGeeks

Given a book of words. Assume you have enough main memory to accommodate all words. design a data structure to find top K maximum occurring words. The data structure should be dynamic so that new words can be added.

A simple solution is to **use Hashing**. Hash all words one by one in a hash table. If a word is already present, then increment its count. Finally, traverse through the hash table and return the k words with maximum counts.

We can **use Trie and Min Heap** to get the k most frequent words efficiently. The idea is to use Trie for searching existing words adding new words efficiently. Trie also stores count of occurrences of words. A Min Heap of size k is used to keep track of k most frequent words at any point of time(Use of Min Heap is same as we used it to find k largest elements in this post).
Trie and Min Heap are linked with each other by storing an additional field in Trie 'indexMinHeap' and a pointer 'trNode' in Min Heap. The value of 'indexMinHeap' is maintained as -1 for the words which are currently not in Min Heap (or currently not among the top k frequent words). For the words which are present in Min Heap, 'indexMinHeap' contains, index of the word in Min Heap. The pointer 'trNode' in Min Heap points to the leaf node corresponding to the word in Trie.

Following is the complete process to print k most frequent words from a file.

Read all words one by one. For every word, insert it into Trie. Increase the counter of the word, if already exists. Now, we need to insert this word in min heap also. For insertion in min heap, 3 cases arise:

1. The word is already present. We just increase the corresponding frequency value in min heap and call minHeapify() for the index obtained by "indexMinHeap" field in Trie. When the min heap nodes are being swapped, we change the corresponding minHeapIndex in the Trie. Remember each node of the min heap is also having pointer to Trie leaf node.

2. The minHeap is not full. we will insert the new word into min heap & update the root node in the min heap node & min heap index in Trie leaf node. Now, call buildMinHeap().

3. The min heap is full. Two sub-cases arise.
....**3.1** The frequency of the new word inserted is less than the frequency of the word stored in the head of min heap. Do nothing.

....**3.2** The frequency of the new word inserted is greater than the frequency of the word stored in the head of min heap. Replace & update the fields. Make sure to update the corresponding min heap index of the "word to be replaced" in Trie with -1 as the word is no longer in min heap.

4. Finally, Min Heap will have the k most frequent words of all words present in given file. So we just need to print all words present in Min Heap.

```
 // A program to find k most frequent words in a file
#include <stdio.h>
#include <string.h>
#include <ctype.h>

# define MAX_CHARS 26
# define MAX_WORD_SIZE 30

// A Trie node
struct TrieNode
{
    bool isEnd; // indicates end of word
    unsigned frequency;  // the number of occurrences of a word
    int indexMinHeap; // the index of the word in minHeap
    TrieNode* child[MAX_CHARS]; // represents 26 slots each for 'a' to 'z'.
};

// A Min Heap node
struct MinHeapNode
{
    TrieNode* root; // indicates the leaf node of TRIE
    unsigned frequency; //  number of occurrences
    char* word; // the actual word stored
};

// A Min Heap
struct MinHeap
{
    unsigned capacity; // the total size a min heap
    int count; // indicates the number of slots filled.
    MinHeapNode* array; //  represents the collection of minHeapNodes
};

// A utility function to create a new Trie node
```

```
TrieNode* newTrieNode()
{
    // Allocate memory for Trie Node
    TrieNode* trieNode = new TrieNode;

    // Initialize values for new node
    trieNode->isEnd = 0;
    trieNode->frequency = 0;
    trieNode->indexMinHeap = -1;
    for( int i = 0; i < MAX_CHARS; ++i )
        trieNode->child[i] = NULL;

    return trieNode;
}

// A utility function to create a Min Heap of given capacity
MinHeap* createMinHeap( int capacity )
{
    MinHeap* minHeap = new MinHeap;

    minHeap->capacity = capacity;
    minHeap->count  = 0;

    // Allocate memory for array of min heap nodes
    minHeap->array = new MinHeapNode [ minHeap->capacity ];

    return minHeap;
}

// A utility function to swap two min heap nodes. This function
// is needed in minHeapify
void swapMinHeapNodes ( MinHeapNode* a, MinHeapNode* b )
{
    MinHeapNode temp = *a;
    *a = *b;
    *b = temp;
}

// This is the standard minHeapify function. It does one thing extra.
// It updates the minHapIndex in Trie when two nodes are swapped in
// in min heap
void minHeapify( MinHeap* minHeap, int idx )
{
    int left, right, smallest;

    left = 2 * idx + 1;
    right = 2 * idx + 2;
    smallest = idx;
```

```
    if ( left < minHeap->count &&
         minHeap->array[ left ]. frequency <
         minHeap->array[ smallest ]. frequency
       )
        smallest = left;

    if ( right < minHeap->count &&
         minHeap->array[ right ]. frequency <
         minHeap->array[ smallest ]. frequency
       )
        smallest = right;

    if( smallest != idx )
    {
        // Update the corresponding index in Trie node.
        minHeap->array[ smallest ]. root->indexMinHeap = idx;
        minHeap->array[ idx ]. root->indexMinHeap = smallest;

        // Swap nodes in min heap
        swapMinHeapNodes (&minHeap->array[ smallest ], &minHeap->array[ idx ]);

        minHeapify( minHeap, smallest );
    }
}

// A standard function to build a heap
void buildMinHeap( MinHeap* minHeap )
{
    int n, i;
    n = minHeap->count - 1;

    for( i = ( n - 1 ) / 2; i >= 0; --i )
        minHeapify( minHeap, i );
}

// Inserts a word to heap, the function handles the 3 cases explained above
void insertInMinHeap( MinHeap* minHeap, TrieNode** root, const char* word )
{
    // Case 1: the word is already present in minHeap
    if( (*root)->indexMinHeap != -1 )
    {
        ++( minHeap->array[ (*root)->indexMinHeap ]. frequency );

        // percolate down
        minHeapify( minHeap, (*root)->indexMinHeap );
    }

    // Case 2: Word is not present and heap is not full
```

```
    else if( minHeap->count < minHeap->capacity )
    {
        int count = minHeap->count;
        minHeap->array[ count ]. frequency = (*root)->frequency;
        minHeap->array[ count ]. word = new char [strlen( word ) + 1];
        strcpy( minHeap->array[ count ]. word, word );

        minHeap->array[ count ]. root = *root;
        (*root)->indexMinHeap = minHeap->count;

        ++( minHeap->count );
        buildMinHeap( minHeap );
    }

    // Case 3: Word is not present and heap is full. And frequency of word
    // is more than root. The root is the least frequent word in heap,
    // replace root with new word
    else if ( (*root)->frequency > minHeap->array[0]. frequency )
    {

        minHeap->array[ 0 ]. root->indexMinHeap = -1;
        minHeap->array[ 0 ]. root = *root;
        minHeap->array[ 0 ]. root->indexMinHeap = 0;
        minHeap->array[ 0 ]. frequency = (*root)->frequency;

        // delete previously allocated memoory and
        delete [] minHeap->array[ 0 ]. word;
        minHeap->array[ 0 ]. word = new char [strlen( word ) + 1];
        strcpy( minHeap->array[ 0 ]. word, word );

        minHeapify ( minHeap, 0 );
    }
}

// Inserts a new word to both Trie and Heap
void insertUtil ( TrieNode** root, MinHeap* minHeap,
                        const char* word, const char* dupWord )
{
    // Base Case
    if ( *root == NULL )
        *root = newTrieNode();

    //  There are still more characters in word
    if ( *word != '\0' )
        insertUtil ( &((*root)->child[ tolower( *word ) - 97 ]),
                         minHeap, word + 1, dupWord );
    else // The complete word is processed
    {
```

```
        // word is already present, increase the frequency
        if ( (*root)->isEnd )
            ++( (*root)->frequency );
        else
        {
            (*root)->isEnd = 1;
            (*root)->frequency = 1;
        }

        // Insert in min heap also
        insertInMinHeap( minHeap, root, dupWord );
    }
}


// add a word to Trie & min heap.  A wrapper over the insertUtil
void insertTrieAndHeap(const char *word, TrieNode** root, MinHeap* minHeap)
{
    insertUtil( root, minHeap, word, word );
}

// A utility function to show results, The min heap
// contains k most frequent words so far, at any time
void displayMinHeap( MinHeap* minHeap )
{
    int i;

    // print top K word with frequency
    for( i = 0; i < minHeap->count; ++i )
    {
        printf( "%s : %d\n", minHeap->array[i].word,
                             minHeap->array[i].frequency );
    }
}

// The main funtion that takes a file as input, add words to heap
// and Trie, finally shows result from heap
void printKMostFreq( FILE* fp, int k )
{
    // Create a Min Heap of Size k
    MinHeap* minHeap = createMinHeap( k );

    // Create an empty Trie
    TrieNode* root = NULL;

    // A buffer to store one word at a time
    char buffer[MAX_WORD_SIZE];
```

```
    // Read words one by one from file.  Insert the word in Trie and Min Heap
    while( fscanf( fp, "%s", buffer ) != EOF )
        insertTrieAndHeap(buffer, &root, minHeap);

    // The Min Heap will have the k most frequent words, so print Min Heap nodes
    displayMinHeap( minHeap );
}

// Driver program to test above functions
int main()
{
    int k = 5;
    FILE *fp = fopen ("file.txt", "r");
    if (fp == NULL)
        printf ("File doesn't exist ");
    else
        printKMostFreq (fp, k);
    return 0;
}
```

Output:

```
your : 3
well : 3
and : 4
to : 4
Geeks : 6
```

The above output is for a file with following content.

```
Welcome to the world of Geeks
This portal has been created to provide well written well thought and well explained
solutions for selected questions If you like Geeks for Geeks and would like to contribute
here is your chance You can write article and mail your article to contribute at
geeksforgeeks org See your article appearing on the Geeks for Geeks main page and help
thousands of other Geeks
```

Source

https://www.geeksforgeeks.org/find-the-k-most-frequent-words-from-a-file/

Chapter 92

Find the largest pair sum in an unsorted array

Find the largest pair sum in an unsorted array - GeeksforGeeks

Given an unsorted of distinct integers, find the largest pair sum in it. For example, the largest pair sum in {12, 34, 10, 6, 40} is 74.

Difficulty Level: Rookie
Expected Time Complexity: O(n) [Only one traversal of array is allowed]

This problem mainly boils down to finding the largest and second largest element in array. We can find the largest and second largest in O(n) time by traversing array once.

```
1) Initialize both first and second largest
   first = max(arr[0], arr[1])
   second = min(arr[0], arr[1])
2) Loop through remaining elements (from 3rd to end)
   a) If the current element is greater than first, then update first
       and second.
   b) Else if the current element is greater than second then update
    second
3) Return (first + second)
```

Below is the implementation of above algorithm:

C++

```
 // C++ program to find largest pair sum in a given array
#include<iostream>
using namespace std;
```

```
/* Function to return largest pair sum. Assumes that
   there are at-least  two elements in arr[] */
int findLargestSumPair(int arr[], int n)
{
    // Initialize first and second largest element
    int first, second;
    if (arr[0] > arr[1])
    {
        first = arr[0];
        second = arr[1];
    }
    else
    {
        first = arr[1];
        second = arr[0];
    }

    // Traverse remaining array and find first and second largest
    // elements in overall array
    for (int i = 2; i<n; i ++)
    {
        /* If current element is greater than first then update both
          first and second */
        if (arr[i] > first)
        {
            second = first;
            first = arr[i];
        }

        /* If arr[i] is in between first and second then update second  */
        else if (arr[i] > second && arr[i] != first)
            second = arr[i];
    }
    return (first + second);
}


/* Driver program to test above function */
int main()
{
    int arr[] = {12, 34, 10, 6, 40};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Max Pair Sum is " << findLargestSumPair(arr, n);
    return 0;
}
```

Java

```
 // Java program to find largest pair sum in a given array

class Test
{
    static int arr[] = new int[]{12, 34, 10, 6, 40};

    /* Method to return largest pair sum. Assumes that
       there are at-least  two elements in arr[] */
    static int findLargestSumPair()
    {
        // Initialize first and second largest element
        int first, second;
        if (arr[0] > arr[1])
        {
            first = arr[0];
            second = arr[1];
        }
        else
        {
            first = arr[1];
            second = arr[0];
        }

        // Traverse remaining array and find first and second largest
        // elements in overall array
        for (int i = 2; i<arr.length; i ++)
        {
            /* If current element is greater than first then update both
              first and second */
            if (arr[i] > first)
            {
                second = first;
                first = arr[i];
            }

            /* If arr[i] is in between first and second then update second  */
            else if (arr[i] > second && arr[i] != first)
                second = arr[i];
        }
        return (first + second);
    }
    // Driver method to test the above function
    public static void main(String[] args)
    {

        System.out.println("Max Pair Sum is " + findLargestSumPair());

    }
```

```
}
```

Python3

```
 # Python3 program to find largest
# pair sum in a given array

# Function to return largest pair
# sum. Assumes that there are
# at-least two elements in arr[]
def findLargestSumPair(arr, n):

    # Initialize first and second
    # largest element
    if arr[0] > arr[1]:
        first = arr[0]
        second = arr[1]

    else:
        first = arr[1]
        second = arr[0]


    # Traverse remaining array and
    # find first and second largest
    # elements in overall array
    for i in range(2, n):

        # If current element is greater
        # than first then update both
        # first and second
        if arr[i] > first:
            second = first
            first = arr[i]


        # If arr[i] is in between first
        # and second then update second
        elif arr[i] > second and arr[i] != first:
            second = arr[i]

    return (first + second)

# Driver program to test above function */
arr = [12, 34, 10, 6, 40]
n = len(arr)
print("Max Pair Sum is",
      findLargestSumPair(arr, n))
```

```
# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to find largest
// pair sum in a given array
using System;

class GFG
{
    /* Method to return largest pair
    sum. Assumes that there are
    at-least two elements in arr[] */
    static int findLargestSumPair(int []arr)
    {
        // Initialize first and
        // second largest element
        int first, second;
        if (arr[0] > arr[1])
        {
            first = arr[0];
            second = arr[1];
        }
        else
        {
            first = arr[1];
            second = arr[0];
        }

        // Traverse remaining array and
        // find first and second largest
        // elements in overall array
        for (int i = 2; i < arr.Length; i ++)
        {
            /* If current element is greater
               than first then update both
               first and second */
            if (arr[i] > first)
            {
                second = first;
                first = arr[i];
            }

            /* If arr[i] is in between first
               and second then update second */
            else if (arr[i] > second &&
                     arr[i] != first)
```

```
                second = arr[i];
        }
        return (first + second);
    }
    // Driver Code
    public static void Main()
    {
        int []arr1 = new int[]{12, 34, 10, 6, 40};
        Console.Write("Max Pair Sum is " +
                      findLargestSumPair(arr1));

    }
}
```

PHP

```
 <?php
// PHP program to find largest
// pair sum in a given array

// Function to return largest
// pair sum. Assumes that
// there are at-least two
// elements in arr[] */
function findLargestSumPair($arr, $n)
{

    // Initialize first and
    // second largest element
    $first;
    $second;

    if ($arr[0] > $arr[1])
    {
        $first = $arr[0];
        $second = $arr[1];
    }
    else
    {
        $first = $arr[1];
        $second = $arr[0];
    }

    // Traverse remaining array
    // and find first and second
    // largest elements in overall
    // array
    for ( $i = 2; $i<$n; $i ++)
```

```
    {

        // If current element is greater
        // than first then update both
        // first and second
        if ($arr[$i] > $first)
        {
            $second = $first;
            $first = $arr[$i];
        }

        // If arr[i] is in between first
        // and second then update second
        else if ($arr[$i] > $second and
                 $arr[$i] != $first)
            $second = $arr[$i];
    }
    return ($first + $second);
}

    // Driver Code
    $arr = array(12, 34, 10, 6, 40);
    $n = count($arr);
    echo "Max Pair Sum is "
          , findLargestSumPair($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
Max Pair Sum is 74
```

Time complexity of above solution is O(n).

This article is contributed by **Rishabh**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/find-the-largest-pair-sum-in-an-unsorted-array/

Chapter 93

Find the largest three elements in an array

Find the largest three elements in an array - GeeksforGeeks

Given an array with all distinct elements, find the largest three elements. Expected time complexity is O(n) and extra space is O(1).

Examples :

```
Input: arr[] = {10, 4, 3, 50, 23, 90}
Output: 90, 50, 23
```

Below is algorithm:

```
1) Initialize the largest three elements as minus infinite.
    first = second = third = -∞

2) Iterate through all elements of array.
   a) Let current array element be x.
   b) If (x > first)
      {
          // This order of assignment is important
          third = second
          second = first
          first = x
       }
   c)  Else if (x > second)
      {
          third = second
```

```
          second = x
      }
   d)  Else if (x > third)
      {
          third = x
      }

3) Print first, second and third.
```

Below is the implementation of above algorithm.

C

```
 #include <stdio.h>
#include <limits.h> /* For INT_MIN */

/* Function to print three largest elements */
void print2largest(int arr[], int arr_size)
{
    int i, first, second, third;

    /* There should be atleast two elements */
    if (arr_size < 3)
    {
        printf(" Invalid Input ");
        return;
    }

    third = first = second = INT_MIN;
    for (i = 0; i < arr_size ; i ++)
    {
        /* If current element is smaller than first*/
        if (arr[i] > first)
        {
            third = second;
            second = first;
            first = arr[i];
        }

        /* If arr[i] is in between first and second then update second  */
        else if (arr[i] > second)
        {
            third = second;
            second = arr[i];
        }

        else if (arr[i] > third)
```

```
            third = arr[i];
    }

    printf("Three largest elements are %d %d %d\n", first, second, third);
}


/* Driver program to test above function */
int main()
{
    int arr[] = {12, 13, 1, 10, 34, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    print2largest(arr, n);
    return 0;
}
```

Java

```
 // Java code to find largest three elements
// in an array

class PrintLargest
{
    /* Function to print three largest elements */
    static void print2largest(int arr[], int arr_size)
    {
        int i, first, second, third;

        /* There should be atleast two elements */
        if (arr_size < 3)
        {
            System.out.print(" Invalid Input ");
            return;
        }

        third = first = second = Integer.MIN_VALUE;
        for (i = 0; i < arr_size ; i ++)
        {
            /* If current element is smaller than
            first*/
            if (arr[i] > first)
            {
                third = second;
                second = first;
                first = arr[i];
            }

            /* If arr[i] is in between first and
```

```
            second then update second  */
            else if (arr[i] > second)
            {
                third = second;
                second = arr[i];
            }

            else if (arr[i] > third)
                third = arr[i];
        }

        System.out.println("Three largest elements are " +
                        first + " " + second + " " + third);
    }

    /* Driver program to test above function*/
    public static void main (String[] args)
    {
        int arr[] = {12, 13, 1, 10, 34, 1};
        int n = arr.length;
        print2largest(arr, n);
    }
}
/*This code is contributed by Prakriti Gupta*/
```

Python3

```
 # Python3 code to find largest three
# elements in an array
import sys

# Function to print three largest
# elements
def print2largest(arr, arr_size):

    # There should be atleast two
    # elements
    if (arr_size < 3):

        print(" Invalid Input ")
        return

    third = first = second = -sys.maxsize

    for i in range(0, arr_size):

        # If current element is smaller
        # than first
```

```
        if (arr[i] > first):

            third = second
            second = first
            first = arr[i]


        # If arr[i] is in between first
        # and second then update second
        elif (arr[i] > second):

            third = second
            second = arr[i]

        elif (arr[i] > third):
            third = arr[i]

    print("Three largest elements are",
                  first, second, third)

# Driver program to test above function
arr = [12, 13, 1, 10, 34, 1]
n = len(arr)
print2largest(arr, n)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# code to find largest
// three elements in an array
using System;

class PrintLargest
{

    // Function to print three
    // largest elements
    static void print2largest(int[] arr,
                              int arr_size)
    {
        int i, first, second, third;

        // There should be atleast two elements
        if (arr_size < 3)
        {
            Console.WriteLine("Invalid Input");
            return;
```

```
        }

        third = first = second = 000;
        for (i = 0; i < arr_size ; i ++)
        {
            // If current element is
            // smaller than first
            if (arr[i] > first)
            {
                third = second;
                second = first;
                first = arr[i];
            }

            // If arr[i] is in between first
            // and second then update second
            else if (arr[i] > second)
            {
                third = second;
                second = arr[i];
            }

            else if (arr[i] > third)
                third = arr[i];
        }

        Console.WriteLine("Three largest elements are " +
                          first + " " + second +
                          " " + third);
    }

    // Driver code
    public static void Main ()
    {
        int[] arr = new int[] {12, 13, 1, 10, 34, 1};
        int n = arr.Length;
        print2largest(arr, n);
    }
}

// This code is contributed by KRV.
```

PHP

```
 <?php
// PHP code to find largest
// three elements in an array
```

```
// Function to print
// three largest elements
function print2largest($arr, $arr_size)
{
    $i; $first; $second; $third;

    // There should be atleast
    // two elements
    if ($arr_size < 3)
    {
        echo " Invalid Input ";
        return;
    }

    $third = $first = $second = PHP_INT_MIN;
    for ($i = 0; $i < $arr_size ; $i ++)
    {
        // If current element is
        // smaller than first
        if ($arr[$i] > $first)
        {
            $third = $second;
            $second = $first;
            $first = $arr[$i];
        }

        // If arr[i] is in between first
        // and second then update second
        else if ($arr[$i] > $second)
        {
            $third = $second;
            $second = $arr[$i];
        }

        else if ($arr[$i] > $third)
            $third = $arr[$i];
    }

    echo "Three largest elements are ",
       $first," ", $second," ", $third;
}


// Driver Code
$arr = array(12, 13, 1,
             10, 34, 1);
$n = count($arr);
print2largest($arr, $n);
```

```
// This code is contributed by anuj_67.
?>
```

Output :

```
Three largest elements are 34 13 12
```

Improved By : KRV, vt_m

Source

https://www.geeksforgeeks.org/find-the-largest-three-elements-in-an-array/

Chapter 94

Find the maximum element in an array which is first increasing and then decreasing

Find the maximum element in an array which is first increasing and then decreasing - GeeksforGeeks

Given an array of integers which is initially increasing and then decreasing, find the maximum value in the array.
Examples :

```
Input: arr[] = {8, 10, 20, 80, 100, 200, 400, 500, 3, 2, 1}
Output: 500

Input: arr[] = {1, 3, 50, 10, 9, 7, 6}
Output: 50

Corner case (No decreasing part)
Input: arr[] = {10, 20, 30, 40, 50}
Output: 50

Corner case (No increasing part)
Input: arr[] = {120, 100, 80, 20, 0}
Output: 120
```

Method 1 (Linear Search)
We can traverse the array and keep track of maximum and element. And finally return the maximum element.

C

```
 #include <stdio.h>

int findMaximum(int arr[], int low, int high)
{
   int max = arr[low];
   int i;
   for (i = low; i <= high; i++)
   {
       if (arr[i] > max)
          max = arr[i];
   }
   return max;
}

/* Driver program to check above functions */
int main()
{
   int arr[] = {1, 30, 40, 50, 60, 70, 23, 20};
   int n = sizeof(arr)/sizeof(arr[0]);
   printf("The maximum element is %d", findMaximum(arr, 0, n-1));
   getchar();
   return 0;
}
```

Java

```
 // java program to find maximum
// element

class Main
{
    // function to find the
    // maximum element
    static int findMaximum(int arr[], int low, int high)
    {
       int max = arr[low];
       int i;
       for (i = low; i <= high; i++)
       {
           if (arr[i] > max)
              max = arr[i];
       }
       return max;
    }

    // main function
    public static void main (String[] args)
    {
```

```
        int arr[] = {1, 30, 40, 50, 60, 70, 23, 20};
        int n = arr.length;
        System.out.println("The maximum element is "+
                           findMaximum(arr, 0, n-1));
    }
}
```

Python3

```
 # Python3 program to find
# maximum element

def findMaximum(arr, low, high):
    max = arr[low]
    i = low
    for i in range(high+1):
        if arr[i] > max:
            max = arr[i]
    return max

# Driver program to check above functions */
arr = [1, 30, 40, 50, 60, 70, 23, 20]
n = len(arr)
print ("The maximum element is %d"%
        findMaximum(arr, 0, n-1))

# This code is contributed by Shreyanshi Arun.
```

C#

```
 // C# program to find maximum
// element
using System;

class GFG
{
    // function to find the
    // maximum element
    static int findMaximum(int []arr, int low, int high)
    {
        int max = arr[low];
        int i;
        for (i = low; i <= high; i++)
        {
            if (arr[i] > max)
                max = arr[i];
        }
```

```
        return max;
    }

    // Driver code
    public static void Main ()
    {
        int []arr = {1, 30, 40, 50, 60, 70, 23, 20};
        int n = arr.Length;
        Console.Write("The maximum element is "+
                        findMaximum(arr, 0, n-1));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to Find the maximum
// element in an array which is first
// increasing and then decreasing

function findMaximum($arr, $low, $high)
{
$max = $arr[$low];
$i;
for ($i = $low; $i <= $high; $i++)
{
    if ($arr[$i] > $max)
        $max = $arr[$i];
}
return $max;
}

// Driver Code
$arr = array(1, 30, 40, 50,
             60, 70, 23, 20);
$n = count($arr);
echo "The maximum element is ",
      findMaximum($arr, 0, $n-1);

// This code is contributed by anuj_67.
?>
```

Output :

```
The maximum element is 70
```

Time Complexity : O(n)

Method 2 (Binary Search)
We can modify the standard Binary Search algorithm for the given type of arrays.
i) If the mid element is greater than both of its adjacent elements, then mid is the maximum.
ii) If mid element is greater than its next element and smaller than the previous element then maximum lies on left side of mid. Example array: {3, 50, 10, 9, 7, 6}
iii) If mid element is smaller than its next element and greater than the previous element then maximum lies on right side of mid. Example array: {2, 4, 6, 8, 10, 3, 1}

C

```
 #include <stdio.h>

int findMaximum(int arr[], int low, int high)
{

   /* Base Case: Only one element is present in arr[low..high]*/
   if (low == high)
     return arr[low];

   /* If there are two elements and first is greater then
      the first element is maximum */
   if ((high == low + 1) && arr[low] >= arr[high])
      return arr[low];

   /* If there are two elements and second is greater then
      the second element is maximum */
   if ((high == low + 1) && arr[low] < arr[high])
      return arr[high];

   int mid = (low + high)/2;   /*low + (high - low)/2;*/

   /* If we reach a point where arr[mid] is greater than both of
     its adjacent elements arr[mid-1] and arr[mid+1], then arr[mid]
     is the maximum element*/
   if ( arr[mid] > arr[mid + 1] && arr[mid] > arr[mid - 1])
      return arr[mid];

   /* If arr[mid] is greater than the next element and smaller than the previous
    element then maximum lies on left side of mid */
   if (arr[mid] > arr[mid + 1] && arr[mid] < arr[mid - 1])
     return findMaximum(arr, low, mid-1);
   else // when arr[mid] is greater than arr[mid-1] and smaller than arr[mid+1]
     return findMaximum(arr, mid + 1, high);
}

/* Driver program to check above functions */
int main()
```

```
{
   int arr[] = {1, 3, 50, 10, 9, 7, 6};
   int n = sizeof(arr)/sizeof(arr[0]);
   printf("The maximum element is %d", findMaximum(arr, 0, n-1));
   getchar();
   return 0;
}
```

Java

```
 // java program to find maximum
// element

class Main
{
    // function to find the
    // maximum element
    static int findMaximum(int arr[], int low, int high)
    {

       /* Base Case: Only one element is
          present in arr[low..high]*/
       if (low == high)
         return arr[low];

       /* If there are two elements and
          first is greater then the first
          element is maximum */
       if ((high == low + 1) && arr[low] >= arr[high])
          return arr[low];

       /* If there are two elements and
          second is greater then the second
          element is maximum */
       if ((high == low + 1) && arr[low] < arr[high])
          return arr[high];

       /*low + (high - low)/2;*/
       int mid = (low + high)/2;

       /* If we reach a point where arr[mid]
          is greater than both of its adjacent
          elements arr[mid-1] and arr[mid+1],
          then arr[mid] is the maximum element*/
       if ( arr[mid] > arr[mid + 1] && arr[mid] > arr[mid - 1])
          return arr[mid];

       /* If arr[mid] is greater than the next
```

```
          element and smaller than the previous
          element then maximum lies on left side
          of mid */
       if (arr[mid] > arr[mid + 1] && arr[mid] < arr[mid - 1])
         return findMaximum(arr, low, mid-1);
       else
         return findMaximum(arr, mid + 1, high);
    }

    // main function
    public static void main (String[] args)
    {
        int arr[] = {1, 3, 50, 10, 9, 7, 6};
        int n = arr.length;
        System.out.println("The maximum element is "+
                           findMaximum(arr, 0, n-1));
    }
}
```

Python3

```
 def findMaximum(arr, low, high):
    # Base Case: Only one element is present in arr[low..high]*/
    if low == high:
        return arr[low]

    # If there are two elements and first is greater then
    # the first element is maximum */
    if high == low + 1 and arr[low] >= arr[high]:
        return arr[low];

    # If there are two elements and second is greater then
    # the second element is maximum */
    if high == low + 1 and arr[low] < arr[high]:
        return arr[high]

    mid = (low + high)//2   #low + (high - low)/2;*/

    # If we reach a point where arr[mid] is greater than both of
    # its adjacent elements arr[mid-1] and arr[mid+1], then arr[mid]
    # is the maximum element*/
    if arr[mid] > arr[mid + 1] and arr[mid] > arr[mid - 1]:
        return arr[mid]

    # If arr[mid] is greater than the next element and smaller than the previous
    # element then maximum lies on left side of mid */
    if arr[mid] > arr[mid + 1] and arr[mid] < arr[mid - 1]:
        return findMaximum(arr, low, mid-1)
```

```
    else: # when arr[mid] is greater than arr[mid-1] and smaller than arr[mid+1]
        return findMaximum(arr, mid + 1, high)

# Driver program to check above functions */
arr = [1, 3, 50, 10, 9, 7, 6]
n = len(arr)
print ("The maximum element is %d"% findMaximum(arr, 0, n-1))

# This code is contributed by Shreyanshi Arun.
```

C#

```
 // C# program to find maximum
// element
using System;

class GFG
{
    // function to find the
    // maximum element
    static int findMaximum(int []arr, int low, int high)
    {

    /* Base Case: Only one element is
        present in arr[low..high]*/
    if (low == high)
        return arr[low];

    /* If there are two elements and
        first is greater then the first
        element is maximum */
    if ((high == low + 1) && arr[low] >= arr[high])
        return arr[low];

    /* If there are two elements and
        second is greater then the second
        element is maximum */
    if ((high == low + 1) && arr[low] < arr[high])
        return arr[high];

    /*low + (high - low)/2;*/
    int mid = (low + high)/2;

    /* If we reach a point where arr[mid]
        is greater than both of its adjacent
        elements arr[mid-1] and arr[mid+1],
        then arr[mid] is the maximum element*/
    if ( arr[mid] > arr[mid + 1] && arr[mid] > arr[mid - 1])
```

```
        return arr[mid];

    /* If arr[mid] is greater than the next
        element and smaller than the previous
        element then maximum lies on left side
        of mid */
    if (arr[mid] > arr[mid + 1] && arr[mid] < arr[mid - 1])
        return findMaximum(arr, low, mid-1);
    else
        return findMaximum(arr, mid + 1, high);
    }

    // main function
    public static void Main()
    {
        int []arr = {1, 3, 50, 10, 9, 7, 6};
        int n = arr.Length;
        Console.Write("The maximum element is "+
                            findMaximum(arr, 0, n-1));
    }
}
// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to Find the maximum
// element in an array which is
// first increasing and then decreasing

function findMaximum($arr, $low, $high)
{

    /* Base Case: Only one element
       is present in arr[low..high]*/
    if ($low == $high)
        return $arr[$low];

    /* If there are two elements
       and first is greater then
       the first element is maximum */
    if (($high == $low + 1) &&
        $arr[$low] >= $arr[$high])
        return $arr[$low];

    /* If there are two elements
       and second is greater then
       the second element is maximum */
```

```
    if (($high == $low + 1) &&
         $arr[$low] < $arr[$high])
        return $arr[$high];

    /*low + (high - low)/2;*/
    $mid = ($low + $high) / 2;

    /* If we reach a point where
       arr[mid] is greater than
       both of its adjacent elements
       arr[mid-1] and arr[mid+1],
       then arr[mid] is the maximum
       element */
    if ( $arr[$mid] > $arr[$mid + 1] &&
         $arr[$mid] > $arr[$mid - 1])
        return $arr[$mid];

    /* If arr[mid] is greater than
       the next element and smaller
       than the previous element then
       maximum lies on left side of mid */
    if ($arr[$mid] > $arr[$mid + 1] &&
        $arr[$mid] < $arr[$mid - 1])
        return findMaximum($arr, $low, $mid - 1);

    // when arr[mid] is greater than
    // arr[mid-1] and smaller than
    // arr[mid+1]
    else
        return findMaximum($arr,
                           $mid + 1, $high);
}

// Driver Code
$arr = array(1, 3, 50, 10, 9, 7, 6);
$n = sizeof($arr);
echo("The maximum element is ");
echo(findMaximum($arr, 0, $n-1));

// This code is contributed by nitin mittal.
?>
```

Output :

```
The maximum element is 50
```

Time Complexity : O(Logn)

This method works only for distinct numbers. For example, it will not work for an array like {0, 1, 1, 2, 2, 2, 2, 2, 3, 4, 4, 5, 3, 3, 2, 2, 1, 1}.

Improved By : vt_m, nitin mittal

Source

https://www.geeksforgeeks.org/find-the-maximum-element-in-an-array-which-is-first-increasing-and-then-decreasi

Chapter 95

Find the minimum element in a sorted and rotated array

Find the minimum element in a sorted and rotated array - GeeksforGeeks

A sorted array is rotated at some unknown point, find the minimum element in it.

Following solution assumes that all elements are distinct.

Examples:

```
Input: {5, 6, 1, 2, 3, 4}
Output: 1

Input: {1, 2, 3, 4}
Output: 1

Input: {2, 1}
Output: 1
```

A simple solution is to traverse the complete array and find minimum. This solution requires ?(n) time.
We can do it in O(Logn) using Binary Search. If we take a closer look at above examples, we can easily figure out following pattern:

- The minimum element is the only element whose previous is greater than it. If there is no previous element element, then there is no rotation (first element is minimum). We check this condition for middle element by comparing it with (mid-1)'th and (mid+1)'th elements.
- If minimum element is not at middle (neither mid nor mid + 1), then minimum element lies in either left half or right half.

1. If middle element is smaller than last element, then the minimum element lies in left half
2. Else minimum element lies in right half.

We strongly recommend you to try it yourself before seeing the following implementation.

C/C++

```
 // C program to find minimum element in a sorted and rotated array
#include <stdio.h>

int findMin(int arr[], int low, int high)
{
    // This condition is needed to handle the case when array is not
    // rotated at all
    if (high < low)  return arr[0];

    // If there is only one element left
    if (high == low) return arr[low];

    // Find mid
    int mid = low + (high - low)/2; /*(low + high)/2;*/

    // Check if element (mid+1) is minimum element. Consider
    // the cases like {3, 4, 5, 1, 2}
    if (mid < high && arr[mid+1] < arr[mid])
       return arr[mid+1];

    // Check if mid itself is minimum element
    if (mid > low && arr[mid] < arr[mid - 1])
       return arr[mid];

    // Decide whether we need to go to left half or right half
    if (arr[high] > arr[mid])
       return findMin(arr, low, mid-1);
    return findMin(arr, mid+1, high);
}

// Driver program to test above functions
int main()
{
    int arr1[] =  {5, 6, 1, 2, 3, 4};
    int n1 = sizeof(arr1)/sizeof(arr1[0]);
    printf("The minimum element is %d\n", findMin(arr1, 0, n1-1));

    int arr2[] =  {1, 2, 3, 4};
    int n2 = sizeof(arr2)/sizeof(arr2[0]);
    printf("The minimum element is %d\n", findMin(arr2, 0, n2-1));
```

```
    int arr3[] =  {1};
    int n3 = sizeof(arr3)/sizeof(arr3[0]);
    printf("The minimum element is %d\n", findMin(arr3, 0, n3-1));

    int arr4[] =  {1, 2};
    int n4 = sizeof(arr4)/sizeof(arr4[0]);
    printf("The minimum element is %d\n", findMin(arr4, 0, n4-1));

    int arr5[] =  {2, 1};
    int n5 = sizeof(arr5)/sizeof(arr5[0]);
    printf("The minimum element is %d\n", findMin(arr5, 0, n5-1));

    int arr6[] =  {5, 6, 7, 1, 2, 3, 4};
    int n6 = sizeof(arr6)/sizeof(arr6[0]);
    printf("The minimum element is %d\n", findMin(arr6, 0, n6-1));

    int arr7[] =  {1, 2, 3, 4, 5, 6, 7};
    int n7 = sizeof(arr7)/sizeof(arr7[0]);
    printf("The minimum element is %d\n", findMin(arr7, 0, n7-1));

    int arr8[] =  {2, 3, 4, 5, 6, 7, 8, 1};
    int n8 = sizeof(arr8)/sizeof(arr8[0]);
    printf("The minimum element is %d\n", findMin(arr8, 0, n8-1));

    int arr9[] =  {3, 4, 5, 1, 2};
    int n9 = sizeof(arr9)/sizeof(arr9[0]);
    printf("The minimum element is %d\n", findMin(arr9, 0, n9-1));

    return 0;
}
```

Java

```
 // Java program to find minimum element in a sorted and rotated array
import java.util.*;
import java.lang.*;
import java.io.*;

class Minimum
{
    static int findMin(int arr[], int low, int high)
    {
        // This condition is needed to handle the case when array
        // is not rotated at all
        if (high < low)  return arr[0];

        // If there is only one element left
```

```
    if (high == low) return arr[low];

    // Find mid
    int mid = low + (high - low)/2; /*(low + high)/2;*/

    // Check if element (mid+1) is minimum element. Consider
    // the cases like {3, 4, 5, 1, 2}
    if (mid < high && arr[mid+1] < arr[mid])
        return arr[mid+1];

    // Check if mid itself is minimum element
    if (mid > low && arr[mid] < arr[mid - 1])
        return arr[mid];

    // Decide whether we need to go to left half or right half
    if (arr[high] > arr[mid])
        return findMin(arr, low, mid-1);
    return findMin(arr, mid+1, high);
}

// Driver Program
public static void main (String[] args)
{
    int arr1[] =  {5, 6, 1, 2, 3, 4};
    int n1 = arr1.length;
    System.out.println("The minimum element is "+ findMin(arr1, 0, n1-1));

    int arr2[] =  {1, 2, 3, 4};
    int n2 = arr2.length;
    System.out.println("The minimum element is "+ findMin(arr2, 0, n2-1));

    int arr3[] =  {1};
    int n3 = arr3.length;
    System.out.println("The minimum element is "+ findMin(arr3, 0, n3-1));

    int arr4[] =  {1, 2};
    int n4 = arr4.length;
    System.out.println("The minimum element is "+ findMin(arr4, 0, n4-1));

    int arr5[] =  {2, 1};
    int n5 = arr5.length;
    System.out.println("The minimum element is "+ findMin(arr5, 0, n5-1));

    int arr6[] =  {5, 6, 7, 1, 2, 3, 4};
    int n6 = arr6.length;
    System.out.println("The minimum element is "+ findMin(arr6, 0, n1-1));

    int arr7[] =  {1, 2, 3, 4, 5, 6, 7};
```

```
        int n7 = arr7.length;
        System.out.println("The minimum element is "+ findMin(arr7, 0, n7-1));

        int arr8[] =  {2, 3, 4, 5, 6, 7, 8, 1};
        int n8 = arr8.length;
        System.out.println("The minimum element is "+ findMin(arr8, 0, n8-1));

        int arr9[] =  {3, 4, 5, 1, 2};
        int n9 = arr9.length;
        System.out.println("The minimum element is "+ findMin(arr9, 0, n9-1));
    }
}
```

Python

```
 # Python program to find minimum element
# in a sorted and rotated array

def findMin(arr, low, high):
    # This condition is needed to handle the case when array is not
    # rotated at all
    if high < low:
        return arr[0]

    # If there is only one element left
    if high == low:
        return arr[low]

    # Find mid
    mid = int((low + high)/2)

    # Check if element (mid+1) is minimum element. Consider
    # the cases like [3, 4, 5, 1, 2]
    if mid < high and arr[mid+1] < arr[mid]:
        return arr[mid+1]

    # Check if mid itself is minimum element
    if mid > low and arr[mid] < arr[mid - 1]:
        return arr[mid]

    # Decide whether we need to go to left half or right half
    if arr[high] > arr[mid]:
        return findMin(arr, low, mid-1)
    return findMin(arr, mid+1, high)

# Driver program to test above functions
arr1 = [5, 6, 1, 2, 3, 4]
n1 = len(arr1)
```

```
print("The minimum element is " + str(findMin(arr1, 0, n1-1)))

arr2 = [1, 2, 3, 4]
n2 = len(arr2)
print("The minimum element is " + str(findMin(arr2, 0, n2-1)))

arr3 = [1]
n3 = len(arr3)
print("The minimum element is " + str(findMin(arr3, 0, n3-1)))

arr4 = [1, 2]
n4 = len(arr4)
print("The minimum element is " + str(findMin(arr4, 0, n4-1)))

arr5 = [2, 1]
n5 = len(arr5)
print("The minimum element is " + str(findMin(arr5, 0, n5-1)))

arr6 = [5, 6, 7, 1, 2, 3, 4]
n6 = len(arr6)
print("The minimum element is " + str(findMin(arr6, 0, n6-1)))

arr7 = [1, 2, 3, 4, 5, 6, 7]
n7 = len(arr7)
print("The minimum element is " + str(findMin(arr7, 0, n7-1)))

arr8 = [2, 3, 4, 5, 6, 7, 8, 1]
n8 = len(arr8)
print("The minimum element is " + str(findMin(arr8, 0, n8-1)))

arr9 = [3, 4, 5, 1, 2]
n9 = len(arr9)
print("The minimum element is " + str(findMin(arr9, 0, n9-1)))

# This code is contributed by Pratik Chhajer
```

C#

```
 // C# program to find minimum element
// in a sorted and rotated array
using System;

class Minimum {

    static int findMin(int[] arr, int low, int high)
    {
        // This condition is needed to handle
        // the case when array
```

```
    // is not rotated at all
    if (high < low)
        return arr[0];

    // If there is only one element left
    if (high == low)
        return arr[low];

    // Find mid
    // (low + high)/2
    int mid = low + (high - low) / 2;

    // Check if element (mid+1) is minimum element. Consider
    // the cases like {3, 4, 5, 1, 2}
    if (mid < high && arr[mid + 1] < arr[mid])
        return arr[mid + 1];

    // Check if mid itself is minimum element
    if (mid > low && arr[mid] < arr[mid - 1])
        return arr[mid];

    // Decide whether we need to go to
    // left half or right half
    if (arr[high] > arr[mid])
        return findMin(arr, low, mid - 1);
    return findMin(arr, mid + 1, high);
}

// Driver Program
public static void Main()
{
    int[] arr1 = { 5, 6, 1, 2, 3, 4 };
    int n1 = arr1.Length;
    Console.WriteLine("The minimum element is " +
                      findMin(arr1, 0, n1 - 1));

    int[] arr2 = { 1, 2, 3, 4 };
    int n2 = arr2.Length;
    Console.WriteLine("The minimum element is " +
                      findMin(arr2, 0, n2 - 1));

    int[] arr3 = { 1 };
    int n3 = arr3.Length;
    Console.WriteLine("The minimum element is " +
                      findMin(arr3, 0, n3 - 1));

    int[] arr4 = { 1, 2 };
    int n4 = arr4.Length;
```

```
        Console.WriteLine("The minimum element is " +
                          findMin(arr4, 0, n4 - 1));

        int[] arr5 = { 2, 1 };
        int n5 = arr5.Length;
        Console.WriteLine("The minimum element is " +
                          findMin(arr5, 0, n5 - 1));

        int[] arr6 = { 5, 6, 7, 1, 2, 3, 4 };
        int n6 = arr6.Length;
        Console.WriteLine("The minimum element is " +
                          findMin(arr6, 0, n1 - 1));

        int[] arr7 = { 1, 2, 3, 4, 5, 6, 7 };
        int n7 = arr7.Length;
        Console.WriteLine("The minimum element is " +
                          findMin(arr7, 0, n7 - 1));

        int[] arr8 = { 2, 3, 4, 5, 6, 7, 8, 1 };
        int n8 = arr8.Length;
        Console.WriteLine("The minimum element is " +
                          findMin(arr8, 0, n8 - 1));

        int[] arr9 = { 3, 4, 5, 1, 2 };
        int n9 = arr9.Length;
        Console.WriteLine("The minimum element is " +
                          findMin(arr9, 0, n9 - 1));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find minimum
// element in a sorted and
// rotated array

function findMin($arr, $low,
                 $high)
{
    // This condition is needed
    // to handle the case when
    // array is not rotated at all
    if ($high < $low) return $arr[0];

    // If there is only
```

```
    // one element left
    if ($high == $low) return $arr[$low];

    // Find mid
    $mid = $low + ($high - $low) / 2; /*($low + $high)/2;*/

    // Check if element (mid+1)
    // is minimum element.
    // Consider the cases like
    // (3, 4, 5, 1, 2)
    if ($mid < $high &&
        $arr[$mid + 1] < $arr[$mid])
    return $arr[$mid + 1];

    // Check if mid itself
    // is minimum element
    if ($mid > $low &&
        $arr[$mid] < $arr[$mid - 1])
    return $arr[$mid];

    // Decide whether we need
    // to go to left half or
    // right half
    if ($arr[$high] > $arr[$mid])
    return findMin($arr, $low,
                   $mid - 1);
    return findMin($arr,
                   $mid + 1, $high);
}

// Driver Code
$arr1 = array(5, 6, 1, 2, 3, 4);
$n1 = sizeof($arr1);
echo "The minimum element is " .
    findMin($arr1, 0, $n1 - 1) . "\n";

$arr2 = array(1, 2, 3, 4);
$n2 = sizeof($arr2);
echo "The minimum element is " .
    findMin($arr2, 0, $n2 - 1) . "\n";

$arr3 = array(1);
$n3 = sizeof($arr3);
echo "The minimum element is " .
    findMin($arr3, 0, $n3 - 1) . "\n";

$arr4 = array(1, 2);
$n4 = sizeof($arr4);
```

```
echo "The minimum element is " .
    findMin($arr4, 0, $n4 - 1) . "\n";

$arr5 = array(2, 1);
$n5 = sizeof($arr5);
echo "The minimum element is " .
    findMin($arr5, 0, $n5 - 1) . "\n";

$arr6 = array(5, 6, 7, 1, 2, 3, 4);
$n6 = sizeof($arr6);
echo "The minimum element is " .
    findMin($arr6, 0, $n6 - 1) . "\n";

$arr7 = array(1, 2, 3, 4, 5, 6, 7);
$n7 = sizeof($arr7);
echo "The minimum element is " .
    findMin($arr7, 0, $n7 - 1) . "\n";

$arr8 = array(2, 3, 4, 5, 6, 7, 8, 1);
$n8 = sizeof($arr8);
echo "The minimum element is " .
    findMin($arr8, 0, $n8 - 1) . "\n";

$arr9 = array(3, 4, 5, 1, 2);
$n9 = sizeof($arr9);
echo "The minimum element is " .
    findMin($arr9, 0, $n9 - 1) . "\n";

// This code is contributed by ChitraNayal
?>
```

Output:

```
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
The minimum element is 1
```

How to handle duplicates?
It turned out that duplicates can't be handled in O(Logn) time in all cases. Thanks to Amit Jain for inputs. The special cases that cause problems are like {2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 2} and {2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2}. It doesn't look possible to go to left half

or right half by doing constant number of comparisons at the middle. So the problem with repetition can be solved in O(n) worst case.

This article is contributed by **Abhay Rathi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : ChitraNayal

Source

https://www.geeksforgeeks.org/find-minimum-element-in-a-sorted-and-rotated-array/

Chapter 96

Find the missing number in Arithmetic Progression

Find the missing number in Arithmetic Progression - GeeksforGeeks

Given an array that represents elements of arithmetic progression in order. One element is missing in the progression, find the missing number.

Examples:

```
Input: arr[]  = {2, 4, 8, 10, 12, 14}
Output: 6

Input: arr[]  = {1, 6, 11, 16, 21, 31};
Output: 26
```

A **Simple Solution** is to linearly traverse the array and find the missing number. Time complexity of this solution is O(n).

We can solve this problem **in O(Logn) time** using Binary Search. The idea is to go to the middle element. Check if the difference between middle and next to middle is equal to diff or not, if not then the missing element lies between mid and mid+1. If the middle element is equal to $n/2^{th}$ term in Arithmetic Series (Let n be the number of elements in input array), then missing element lies in right half. Else element lies in left half.

Following is implementation of above idea.

C

```
 // A C program to find the missing number in a given
// arithmetic progression
#include <stdio.h>
```

```
#include <limits.h>

// A binary search based recursive function that returns
// the missing element in arithmetic progression
int findMissingUtil(int arr[], int low, int high, int diff)
{
    // There must be two elements to find the missing
    if (high <= low)
        return INT_MAX;

    // Find index of middle element
    int mid = low + (high - low)/2;

    // The element just after the middle element is missing.
    // The arr[mid+1] must exist, because we return when
    // (low == high) and take floor of (high-low)/2
    if (arr[mid+1] - arr[mid] != diff)
        return (arr[mid] + diff);

    // The element just before mid is missing
    if (mid > 0 && arr[mid] - arr[mid-1] != diff)
        return (arr[mid-1] + diff);

    // If the elements till mid follow AP, then recur
    // for right half
    if (arr[mid] == arr[0] + mid*diff)
        return findMissingUtil(arr, mid+1, high, diff);

    // Else recur for left half
    return findMissingUtil(arr, low, mid-1, diff);
}

// The function uses findMissingUtil() to find the missing
// element in AP.  It assumes that there is exactly one missing
// element and may give incorrect result when there is no missing
// element or more than one missing elements.
// This function also assumes that the difference in AP is an
// integer.
int findMissing(int arr[], int n)
{
    // If exactly one element is missing, then we can find
    // difference of arithmetic progression using following
    // formula.  Example, 2, 4, 6, 10, diff = (10-2)/4 = 2.
    // The assumption in formula is that the difference is
    // an integer.
    int diff = (arr[n-1] - arr[0])/n;

    // Binary search for the missing number using above
```

```
    // calculated diff
    return findMissingUtil(arr, 0, n-1, diff);
}

/* Driver program to check above functions */
int main()
{
    int arr[] = {2, 4, 8, 10, 12, 14};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("The missing element is %d", findMissing(arr, n));
    return 0;
}
```

Java

```
 // A Java program to find
// the missing number in
// a given arithmetic
// progression
import java.io.*;

class GFG
{

// A binary search based
// recursive function that
// returns the missing
// element in arithmetic
// progression
static int findMissingUtil(int arr[], int low,
                           int high, int diff)
{
    // There must be two elements
    // to find the missing
    if (high <= low)
        return Integer.MAX_VALUE;

    // Find index of
    // middle element
    int mid = low + (high - low) / 2;

    // The element just after the
    // middle element is missing.
    // The arr[mid+1] must exist,
    // because we return when
    // (low == high) and take
    // floor of (high-low)/2
    if (arr[mid + 1] - arr[mid] != diff)
```

```
        return (arr[mid] + diff);

    // The element just
    // before mid is missing
    if (mid > 0 && arr[mid] -
                  arr[mid - 1] != diff)
        return (arr[mid - 1] + diff);

    // If the elements till mid follow
    // AP, then recur for right half
    if (arr[mid] == arr[0] + mid * diff)
        return findMissingUtil(arr, mid + 1,
                               high, diff);

    // Else recur for left half
    return findMissingUtil(arr, low, mid - 1, diff);
}

// The function uses findMissingUtil()
// to find the missing element in AP.
// It assumes that there is exactly
// one missing element and may give
// incorrect result when there is no
// missing element or more than one
// missing elements. This function also
// assumes that the difference in AP is
// an integer.
static int findMissing(int arr[], int n)
{
    // If exactly one element is missing,
    // then we can find difference of
    // arithmetic progression using
    // following formula. Example, 2, 4,
    // 6, 10, diff = (10-2)/4 = 2.
    // The assumption in formula is that
    // the difference is an integer.
    int diff = (arr[n - 1] - arr[0]) / n;

    // Binary search for the missing
    // number using above calculated diff
    return findMissingUtil(arr, 0, n - 1, diff);
}

// Driver Code
public static void main (String[] args)
{
    int arr[] = {2, 4, 8, 10, 12, 14};
    int n = arr.length;
```

```
    System.out.println("The missing element is "+
                            findMissing(arr, n));
}
}

// This code is contributed by anuj_67.
```

C#

```
 // A C# program to find
// the missing number in
// a given arithmetic
// progression
using System;

class GFG
{

// A binary search based
// recursive function that
// returns the missing
// element in arithmetic
// progression
static int findMissingUtil(int []arr, int low,
                           int high, int diff)
{
    // There must be two elements
    // to find the missing
    if (high <= low)
        return int.MaxValue;

    // Find index of
    // middle element
    int mid = low + (high -
                     low) / 2;

    // The element just after the
    // middle element is missing.
    // The arr[mid+1] must exist,
    // because we return when
    // (low == high) and take
    // floor of (high-low)/2
    if (arr[mid + 1] -
        arr[mid] != diff)
        return (arr[mid] + diff);

    // The element just
    // before mid is missing
```

```
    if (mid > 0 && arr[mid] -
                   arr[mid - 1] != diff)
        return (arr[mid - 1] + diff);

    // If the elements till mid follow
    // AP, then recur for right half
    if (arr[mid] == arr[0] +
               mid * diff)
        return findMissingUtil(arr, mid + 1,
                               high, diff);

    // Else recur for left half
    return findMissingUtil(arr, low,
                           mid - 1, diff);
}

// The function uses findMissingUtil()
// to find the missing element
// in AP. It assumes that there
// is exactly one missing element
// and may give incorrect result
// when there is no missing element
// or more than one missing elements.
// This function also assumes that
// the difference in AP is an integer.
static int findMissing(int []arr, int n)
{
    // If exactly one element
    // is missing, then we can
    // find difference of arithmetic
    // progression using following
    // formula. Example, 2, 4, 6, 10,
    // diff = (10-2)/4 = 2.The assumption
    // in formula is that the difference
    // is an integer.
    int diff = (arr[n - 1] -
                arr[0]) / n;

    // Binary search for the
    // missing number using
    // above calculated diff
    return findMissingUtil(arr, 0,
                           n - 1, diff);
}

// Driver Code
public static void Main ()
{
```

```
    int []arr = {2, 4, 8,
                 10, 12, 14};
    int n = arr.Length;
    Console.WriteLine("The missing element is "+
                           findMissing(arr, n));
}
}

// This code is contributed by anuj_67.
```

Output:

```
The missing element is 6
```

Exercise:
Solve the same problem for Geometrical Series. What is the time complexity of your solution? What about Fibonacci Series?

This article is contributed by **Harshit Agrawal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/find-missing-number-arithmetic-progression/

Chapter 97

Find the nearest smaller numbers on left side in an array

Find the nearest smaller numbers on left side in an array - GeeksforGeeks

Given an array of integers, find the nearest smaller number for every element such that the smaller element is on left side.

Examples:

```
Input:  arr[] = {1, 6, 4, 10, 2, 5}
Output:         {_, 1, 1,  4, 1, 2}
First element ('1') has no element on left side. For 6,
there is only one smaller element on left side '1'.
For 10, there are three smaller elements on left side (1,
6 and 4), nearest among the three elements is 4.

Input: arr[] = {1, 3, 0, 2, 5}
Output:        {_, 1, _, 0, 2}
```

Expected time complexity is O(n).

A **Simple Solution** is to use two nested loops. The outer loop starts from second element, the inner loop goes to all elements on left side of the element picked by outer loop and stops as soon as it finds a smaller element.

C++

```
 // C++ implementation of simple algorithm to find
// smaller element on left side
#include <iostream>
using namespace std;
```

```
// Prints smaller elements on left side of every element
void printPrevSmaller(int arr[], int n)
{
    // Always print empty or '_' for first element
    cout << "_, ";

    // Start from second element
    for (int i=1; i<n; i++)
    {
        // look for smaller element on left of 'i'
        int j;
        for (j=i-1; j>=0; j--)
        {
            if (arr[j] < arr[i])
            {
                cout << arr[j] << ", ";
                break;
            }
        }

        // If there is no smaller element on left of 'i'
        if (j == -1)
           cout << "_, " ;
    }
}

/* Driver program to test insertion sort */
int main()
{
    int arr[] = {1, 3, 0, 2, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    printPrevSmaller(arr, n);
    return 0;
}
```

Java

```
 // Java implementation of simple
// algorithm to find smaller
// element on left side
import java.io.*;
class GFG {

// Prints smaller elements on
// left side of every element
static void printPrevSmaller(int []arr, int n)
{
```

```
    // Always print empty or '_'
    // for first element
    System.out.print( "_, ");

    // Start from second element
    for (int i = 1; i < n; i++)
    {
        // look for smaller
        // element on left of 'i'
        int j;
        for(j = i - 1; j >= 0; j--)
        {
            if (arr[j] < arr[i])
            {
                System.out.print(arr[j] + ", ");
                break;
            }
        }

        // If there is no smaller
        // element on left of 'i'
        if (j == -1)
        System.out.print( "_, ") ;
    }
}

    // Driver Code
    public static void main (String[] args)
    {
        int []arr = {1, 3, 0, 2, 5};
        int n = arr.length;
        printPrevSmaller(arr, n);
    }
}

// This code is contributed by anuj_67.
```

Python3

```
 # Python 3 implementation of simple
# algorithm to find smaller element
# on left side

# Prints smaller elements on left
# side of every element
def printPrevSmaller(arr, n):
```

```
    # Always print empty or '_' for
    # first element
    print("_, ", end="")

    # Start from second element
    for i in range(1, n ):

        # look for smaller element
        # on left of 'i'
        for j in range(i-1 ,-2 ,-1):

            if (arr[j] < arr[i]):

                print(arr[j] ,", ",
                            end="")
                break

        # If there is no smaller
        # element on left of 'i'
        if (j == -1):
            print("_, ", end="")

# Driver program to test insertion
# sort
arr = [1, 3, 0, 2, 5]
n = len(arr)
printPrevSmaller(arr, n)

# This code is contributed by
# Smitha
```

C#

```
 // C# implementation of simple
// algorithm to find smaller
// element on left side
using System;

class GFG {

    // Prints smaller elements on
    // left side of every element
    static void printPrevSmaller(int []arr,
                                 int n)
    {

        // Always print empty or '_'
        // for first element
```

```
        Console.Write( "_, ");

        // Start from second element
        for (int i = 1; i < n; i++)
        {
            // look for smaller
            // element on left of 'i'
            int j;
            for(j = i - 1; j >= 0; j--)
            {
                if (arr[j] < arr[i])
                {
                    Console.Write(arr[j]
                                + ", ");
                    break;
                }
            }

            // If there is no smaller
            // element on left of 'i'
            if (j == -1)
            Console.Write( "_, ") ;
        }
    }

    // Driver Code
    public static void Main ()
    {
        int []arr = {1, 3, 0, 2, 5};
        int n = arr.Length;
        printPrevSmaller(arr, n);
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP implementation of simple
// algorithm to find smaller
// element on left side

// Prints smaller elements on
// left side of every element
function printPrevSmaller( $arr, $n)
{
```

```
    // Always print empty or
    // '_' for first element
    echo "_, ";

    // Start from second element
    for($i = 1; $i < $n; $i++)
    {

        // look for smaller
        // element on left of 'i'
        $j;
        for($j = $i - 1; $j >= 0; $j--)
        {
            if ($arr[$j] < $arr[$i])
            {
                echo $arr[$j] , ", ";
                break;
            }
        }

        // If there is no smaller
        // element on left of 'i'
        if ($j == -1)
        echo "_, " ;
    }
}

    // Driver Code
    $arr = array(1, 3, 0, 2, 5);
    $n = count($arr);
    printPrevSmaller($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
_, 1, _, 0, 2, ,
```

Time complexity of the above solution is $O(n^2)$.

There can be an **Efficient Solution** that works in O(n) time. The idea is to use a stack. Stack is used to maintain a subsequence of the values that have been processed so far and are smaller than any later value that has already been processed.

Below is stack based algorithm

```
Let input sequence be 'arr[]' and size of array be 'n'

1) Create a new empty stack S

2) For every element 'arr[i]' in the input sequence 'arr[]',
   where 'i' goes from 0 to n-1.
    a) while S is nonempty and the top element of
       S is greater than or equal to 'arr[i]':
           pop S

    b) if S is empty:
           'arr[i]' has no preceding smaller value
    c) else:
           the nearest smaller value to 'arr[i]' is
           the top element of S

    d) push 'arr[i]' onto S
```

Below is C++ implementation of above algorithm.

```
 // C++ implementation of simple algorithm to find
// smaller element on left side
#include <iostream>
#include <stack>
using namespace std;

// Prints smaller elements on left side of every element
void printPrevSmaller(int arr[], int n)
{
    // Create an empty stack
    stack<int> S;

    // Traverse all array elements
    for (int i=0; i<n; i++)
    {
        // Keep removing top element from S while the top
        // element is greater than or equal to arr[i]
        while (!S.empty() && S.top() >= arr[i])
            S.pop();

        // If all elements in S were greater than arr[i]
        if (S.empty())
            cout << "_, ";
        else  //Else print the nearest smaller element
            cout << S.top() << ", ";

        // Push this element
        S.push(arr[i]);
```

```
    }
}

/* Driver program to test insertion sort */
int main()
{
    int arr[] = {1, 3, 0, 2, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    printPrevSmaller(arr, n);
    return 0;
}
```

Output:

```
_, 1, _, 0, 2,
```

Time complexity of above program is O(n) as every element is pushed and popped at most once to the stack. So overall constant number of operations are performed per element.

This article is contributed by Ashish Kumar Singh. Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-the-nearest-smaller-numbers-on-left-side-in-an-array/

Chapter 98

Find the number of times every day occurs in a month

Find the number of times every day occurs in a month - GeeksforGeeks

Given the starting day and the number of days in a month. Find the number of times every day occurs in a month

Examples:

```
Input : Number of days in month = 28
        First day = Wednesday
Output : Monday = 4
         Tuesday = 4
         Wednesday = 4
         Thursday = 4
         Friday = 4
         Saturday = 4
         Sunday = 4
Explanation: In the month of February,
every day occurs 4 times.

Input : Number of days in  month = 31
        First day = Tuesday
Output : Monday = 4
         Tuesday = 5
         Wednesday = 5
         Thursday = 5
         Friday = 4
         Saturday = 4
         Sunday = 4
Explanation: The month starts on Tuesday
```

and ends on Thursday.

Observations: We have to make some key observations. First one will be if the month has 28 days then every day occurs 4 times. The second one will be if it has 29 days then the day on which the month starts will occur 5 times. The third one will be if the month has 30 days, then the day on which the month starts and the next day will occur 5 days. The last one is if the month has 31 days, then the day on which the month starts and the next 2 days will occur 5 days with the rest occurring 4 times each.

Approach: Create an count array with size 7 and having an initial value of 4 as the minimum number of occurrence will be 4. **(number of days-28)**. Find the index of the firstday. Calculate the number of days whose occurrence will be 5. Then run a loop from pos to pos+(number of days-28) to mark the occurrence as 5. If pos+(number of days-28) exceeds 6, then use %7 to get to the indexes from the beginning.

Below is the C++ implementation of the above approach:

CPP

```
 // C++ program to count occurrence of days in a month
#include <bits/stdc++.h>
using namespace std;

// function to find occurrences
void occurrenceDays(int n, string firstday)
{
    // stores days in a week
    string days[] = { "Monday", "Tuesday", "Wednesday",
           "Thursday",  "Friday", "Saturday", "Sunday" };

    // Initialize all counts as 4.
    int count[7];
    for (int i = 0; i < 7; i++)
        count[i] = 4;

    // find index of the first day
    int pos;
    for (int i = 0; i < 7; i++) {
        if (firstday == days[i]) {
            pos = i;
            break;
        }
    }

    // number of days whose occurrence will be 5
    int inc = n - 28;

    // mark the occurrence to be 5 of n-28 days
```

```
    for (int i = pos; i < pos + inc; i++) {
        if (i > 6)
            count[i % 7] = 5;
        else
            count[i] = 5;
    }

    // print the days
    for (int i = 0; i < 7; i++) {
        cout << days[i] << " " << count[i] << endl;
    }
}

// driver program to test the above fucntion
int main()
{
    int n = 31;
    string firstday = "Tuesday";
    occurrenceDays(n, firstday);
    return 0;
}
```

Java

```
 // Java program to count
// occurrence of days in a month
import java.util.*;
import java.lang.*;

public class GfG{

    // function to find occurrences
    public static void occurrenceDays(int n, String firstday)
    {
        // stores days in a week
        String[] days = new String[]{ "Monday",
                "Tuesday", "Wednesday",
                "Thursday", "Friday",
                "Saturday", "Sunday" };

        // Initialize all counts as 4.
        int[] count = new int[7];
        for (int i = 0; i < 7; i++)
            count[i] = 4;

        // find index of the first day
        int pos = 0;
        for (int i = 0; i < 7; i++)
```

```
        {
            if (firstday == days[i])
            {
                pos = i;
                break;
            }
        }

        // number of days whose occurrence
        // will be 5
        int inc = n - 28;

        // mark the occurrence to be 5 of n-28 days
        for (int i = pos; i < pos + inc; i++)
        {
            if (i > 6)
                count[i % 7] = 5;
            else
                count[i] = 5;
        }

        // print the days
        for (int i = 0; i < 7; i++)
        {
            System.out.println(days[i] + " " + count[i]);
        }
    }

    // Driver function
    public static void main(String argc[]){
        int n = 31;
        String firstday = "Tuesday";
        occurrenceDays(n, firstday);
    }
}

// This code is contributed by Sagar Shukla
```

Python3

```
 # Python program to count
# occurrence of days in a month
import math

# function to find occurrences
def occurrenceDays( n, firstday):

    # stores days in a week
```

```
    days = [ "Monday", "Tuesday", "Wednesday",
           "Thursday",  "Friday", "Saturday",
           "Sunday" ]

    # Initialize all counts as 4.
    count= [4 for i in range(0,7)]

    # find index of the first day
    pos=-1
    for i in range(0,7):
        if (firstday == days[i]):
            pos = i
            break



    # number of days whose occurrence will be 5
    inc = n - 28

    # mark the occurrence to be 5 of n-28 days
    for  i in range( pos, pos + inc):
        if (i > 6):
            count[i % 7] = 5
        else:
            count[i] = 5


    # print the days
    for i in range(0,7):
        print (days[i] , " " , count[i])



# driver program to test
# the above fucntion
n = 31
firstday = "Tuesday"
occurrenceDays(n, firstday)

# This code is contributed by Gitanjali.
```

C#

```
 // C# program to count occurrence of
// days in a month
using System;

public class GfG {
```

```
// function to find occurrences
public static void occurrenceDays(int n,
                          string firstday)
{

    // stores days in a week
    String[] days = new String[]{ "Monday",
                    "Tuesday", "Wednesday",
                      "Thursday", "Friday",
                    "Saturday", "Sunday" };

    // Initialize all counts as 4.
    int[] count = new int[7];

    for (int i = 0; i < 7; i++)
        count[i] = 4;

    // find index of the first day
    int pos = 0;
    for (int i = 0; i < 7; i++)
    {
        if (firstday == days[i])
        {
            pos = i;
            break;
        }
    }

    // number of days whose occurrence
    // will be 5
    int inc = n - 28;

    // mark the occurrence to be 5 of
    // n-28 days
    for (int i = pos; i < pos + inc; i++)
    {
        if (i > 6)
            count[i % 7] = 5;
        else
            count[i] = 5;
    }

    // print the days
    for (int i = 0; i < 7; i++)
    {
        Console.WriteLine(days[i] + " "
                                + count[i]);
```

```
        }
    }

    // Driver function
    public static void Main()
    {
        int n = 31;
        string firstday = "Tuesday";

        occurrenceDays(n, firstday);
    }
}

// This code is contributed by vt_m.
```

Output:

```
Monday 4
Tuesday 5
Wednesday 5
Thursday 5
Friday 4
Saturday 4
Sunday 4
```

Source

https://www.geeksforgeeks.org/find-number-times-every-day-occurs-month/

Chapter 99

Find the number of zeroes

Find the number of zeroes - GeeksforGeeks

Given an array of 1s and 0s which has all 1s first followed by all 0s. Find the number of 0s. Count the number of zeroes in the given array.

Examples :

```
Input: arr[] = {1, 1, 1, 1, 0, 0}
Output: 2

Input: arr[] = {1, 0, 0, 0, 0}
Output: 4

Input: arr[] = {0, 0, 0}
Output: 3

Input: arr[] = {1, 1, 1, 1}
Output: 0
```

A **simple solution** is to traverse the input array. As soon as we find a 0, we return n – index of first 0. Here n is number of elements in input array. Time complexity of this solution would be O(n).

Since the input array is sorted, we can use **Binary Search** to find the first occurrence of 0. Once we have index of first element, we can return count as n – index of first zero.

C

```
 // A divide and conquer solution to find count of zeroes in an array
// where all 1s are present before all 0s
#include <stdio.h>
```

```
/* if 0 is present in arr[] then returns the index of FIRST occurrence
   of 0 in arr[low..high], otherwise returns -1 */
int firstZero(int arr[], int low, int high)
{
    if (high >= low)
    {
        // Check if mid element is first 0
        int mid = low + (high - low)/2;
        if (( mid == 0 || arr[mid-1] == 1) && arr[mid] == 0)
            return mid;

        if (arr[mid] == 1)  // If mid element is not 0
            return firstZero(arr, (mid + 1), high);
        else  // If mid element is 0, but not first 0
            return firstZero(arr, low, (mid -1));
    }
    return -1;
}

// A wrapper over recursive function firstZero()
int countZeroes(int arr[], int n)
{
    // Find index of first zero in given array
    int first = firstZero(arr, 0, n-1);

    // If 0 is not present at all, return 0
    if (first == -1)
        return 0;

    return (n - first);
}

/* Driver program to check above functions */
int main()
{
    int arr[] =   {1, 1, 1, 0, 0, 0, 0, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Count of zeroes is %d", countZeroes(arr, n));
    return 0;
}
```

Java

```
 // A divide and conquer solution to find count of zeroes in an array
// where all 1s are present before all 0s

class CountZeros
{
```

```
    /* if 0 is present in arr[] then returns the index of FIRST occurrence
       of 0 in arr[low..high], otherwise returns -1 */
    int firstZero(int arr[], int low, int high)
    {
        if (high >= low)
        {
            // Check if mid element is first 0
            int mid = low + (high - low) / 2;
            if ((mid == 0 || arr[mid - 1] == 1) && arr[mid] == 0)
                return mid;

            if (arr[mid] == 1) // If mid element is not 0
                return firstZero(arr, (mid + 1), high);
            else // If mid element is 0, but not first 0
                return firstZero(arr, low, (mid - 1));
        }
        return -1;
    }

    // A wrapper over recursive function firstZero()
    int countZeroes(int arr[], int n)
    {
        // Find index of first zero in given array
        int first = firstZero(arr, 0, n - 1);

        // If 0 is not present at all, return 0
        if (first == -1)
            return 0;

        return (n - first);
    }

    // Driver program to test above functions
    public static void main(String[] args)
    {
        CountZeros count = new CountZeros();
        int arr[] = {1, 1, 1, 0, 0, 0, 0, 0};
        int n = arr.length;
        System.out.println("Count of zeroes is " + count.countZeroes(arr, n));
    }
}
```

Python3

```
 # A divide and conquer solution to
# find count of zeroes in an array
# where all 1s are present before all 0s
```

```
# if 0 is present in arr[] then returns
# the index of FIRST occurrence of 0 in
# arr[low..high], otherwise returns -1
def firstZero(arr, low, high):

    if (high >= low):

        # Check if mid element is first 0
        mid = low + int((high - low) / 2)
        if (( mid == 0 or arr[mid-1] == 1)
                      and arr[mid] == 0):
            return mid

        # If mid element is not 0
        if (arr[mid] == 1):
            return firstZero(arr, (mid + 1), high)

        # If mid element is 0, but not first 0
        else:
            return firstZero(arr, low, (mid - 1))

    return -1

# A wrapper over recursive
# function firstZero()
def countZeroes(arr, n):

    # Find index of first zero in given array
    first = firstZero(arr, 0, n - 1)

    # If 0 is not present at all, return 0
    if (first == -1):
        return 0

    return (n - first)

# Driver Code
arr = [1, 1, 1, 0, 0, 0, 0, 0]
n = len(arr)
print("Count of zeroes is",
        countZeroes(arr, n))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // A divide and conquer solution to find
// count of zeroes in an array where all
```

```
// 1s are present before all 0s
using System;

class CountZeros
{
    /* if 0 is present in arr[] then returns
       the index of FIRST occurrence of 0 in
       arr[low..high], otherwise returns -1 */
    int firstZero(int []arr, int low, int high)
    {
        if (high >= low)
        {
            // Check if mid element is first 0
            int mid = low + (high - low) / 2;
            if ((mid == 0 || arr[mid - 1] == 1) &&
                                arr[mid] == 0)
                return mid;

            if (arr[mid] == 1) // If mid element is not 0
                return firstZero(arr, (mid + 1), high);

            else // If mid element is 0, but not first 0
                return firstZero(arr, low, (mid - 1));
        }
        return -1;
    }

    // A wrapper over recursive function firstZero()
    int countZeroes(int []arr, int n)
    {
        // Find index of first zero in given array
        int first = firstZero(arr, 0, n - 1);

        // If 0 is not present at all, return 0
        if (first == -1)
            return 0;

        return (n - first);
    }

    // Driver program to test above functions
    public static void Main()
    {
        CountZeros count = new CountZeros();
        int []arr = {1, 1, 1, 0, 0, 0, 0, 0};
        int n = arr.Length;
        Console.Write("Count of zeroes is " +
                      count.countZeroes(arr, n));
```

```
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// A divide and conquer solution to
// find count of zeroes in an array
// where all 1s are present before all 0s

/* if 0 is present in arr[] then
   returns the index of FIRST
   occurrence of 0 in arr[low..high],
   otherwise returns -1 */
function firstZero($arr, $low, $high)
{
    if ($high >= $low)
    {

        // Check if mid element is first 0
        $mid = $low + floor(($high - $low)/2);

        if (( $mid == 0 || $arr[$mid-1] == 1) &&
                                $arr[$mid] == 0)
            return $mid;

        // If mid element is not 0
        if ($arr[$mid] == 1)
            return firstZero($arr, ($mid + 1), $high);

        // If mid element is 0,
        // but not first 0
        else
            return firstZero($arr, $low,
                            ($mid - 1));
    }
    return -1;
}

// A wrapper over recursive
// function firstZero()
function countZeroes($arr, $n)
{

    // Find index of first
    // zero in given array
```

```
    $first = firstZero($arr, 0, $n - 1);

    // If 0 is not present
    // at all, return 0
    if ($first == -1)
        return 0;

    return ($n - $first);
}

    // Driver Code
    $arr = array(1, 1, 1, 0, 0, 0, 0, 0);
    $n = sizeof($arr);
    echo("Count of zeroes is ");
    echo(countZeroes($arr, $n));

// This code is contributed by nitin mittal
?>
```

Output:

```
Count of zeroes is 5
```

Time Complexity: O(Logn) where n is number of elements in arr[].

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/find-number-zeroes/

Chapter 100

Find the odd appearing element in O(Log n) time

Find the odd appearing element in O(Log n) time - GeeksforGeeks

Given an array where all elements appear even number of times except one. All repeating occurrences of elements appear in pairs and these pairs are not adjacent (there cannot be more than two consecutive occurrences of any element). Find the element that appears odd number of times.

Note that input like {2, 2, 1, 2, 2, 1, 1} is valid as all repeating occurrences occur in pairs and these pairs are not adjacent. Input like {2, 1, 2} is invalid as repeating elements don't appear in pairs. Also, input like {1, 2, 2, 2, 2} is invalid as two pairs of 2 are adjacent. Input like {2, 2, 2, 1} is also invalid as there are three consecutive occurrences of 2.

Example :

```
Input: arr[] = {1, 1, 2, 2, 1, 1, 2, 2, 13, 1, 1, 40, 40, 13, 13}
Output: 13

Input: arr[] = {1, 1, 2, 2, 3, 3, 4, 4, 3, 600, 600, 4, 4}
Output: 3
```

We strongly recommend you to minimize your browser and try this yourself first.

A **Simple Solution** is to sort the array and then traverse the array from left to right. Since the array is sorted, we can easily figure out the required element. Time complexity of this solution is O(n Log n)

A **Better Solution** is to do XOR of all elements, result of XOR would give the odd appearing element. Time complexity of this solution is O(n). See XOR based solution for add apearing for more details.

An **Efficient Solution** can find the required element in O(Log n) time. The idea is to use Binary Search. Below is an observation in input array.
Since the element appears odd number of times, there must be a single occurrence of the element. For example, in {2, 1, 1, 2, 2), the first 2 is the odd occurrence. So the idea is to find this odd occurrence using Binary Search.
All elements before the odd occurrence have first occurrence at even index (0, 2, ..) and next occurrence at odd index (1, 3, …). And all elements afterhave first occurrence at odd index and next occurrence at even index.

1) Find the middle index, say 'mid'.

2) If 'mid' is even, then compare arr[mid] and arr[mid + 1]. If both are same, then there is an odd occurrence of the element after 'mid' else before mid.

3) If 'mid' is odd, then compare arr[mid] and arr[mid – 1]. If both are same, then there is an odd occurrence after 'mid' else before mid.

Below is the implementation based on above idea.

C/C++

```
 // C program to find the element that appears odd number of time
#include<stdio.h>

// A Binary Search based function to find the element
// that appears odd times
void search(int *arr, int low, int high)
{
     // Base cases
    if (low > high)
       return;
    if (low==high)
    {
        printf("The required element is %d ", arr[low]);
        return;
    }

    // Find the middle point
    int mid = (low+high)/2;

    // If mid is even and element next to mid is
    // same as mid, then output element lies on
    // right side, else on left side
    if (mid%2 == 0)
    {
        if (arr[mid] == arr[mid+1])
            search(arr, mid+2, high);
        else
            search(arr, low, mid);
    }
```

```
    else // If mid is odd
    {
        if (arr[mid] == arr[mid-1])
            search(arr, mid+1, high);
        else
            search(arr, low, mid-1);
    }
}

// Driver program
int main()
{
    int arr[] = {1, 1, 2, 2, 1, 1, 2, 2, 13, 1, 1, 40, 40};
    int len = sizeof(arr)/sizeof(arr[0]);
    search(arr, 0, len-1);
    return 0;
}
```

Java

```
 // Java program to find the element
// that appears odd number of time

class GFG
{
    // A Binary Search based function to find
    // the element that appears odd times
    static void search(int arr[], int low, int high)
    {
        // Base cases
        if (low > high)
        return;
        if (low == high)
        {
            System.out.printf("The required element is %d "
                              , arr[low]);
            return;
        }

        // Find the middle point
        int mid = (low + high)/2;

        // If mid is even and element next to mid is
        // same as mid, then output element lies on
        // right side, else on left side
        if (mid % 2 == 0)
        {
            if (arr[mid] == arr[mid + 1])
```

```
                search(arr, mid + 2, high);
            else
                search(arr, low, mid);
        }

        // If mid is odd
        else
        {
            if (arr[mid] == arr[mid - 1])
                search(arr, mid + 1, high);
            else
                search(arr, low, mid - 1);
        }
    }

    // Driver program
    public static void main(String[] args)
    {
        int arr[] = {1, 1, 2, 2, 1, 1, 2, 2, 13,
                                    1, 1, 40, 40};
        int len = arr.length;
        search(arr, 0, len-1);
    }
}
// This code is contributed by
// Smitha DInesh Semwal
```

Python

```
 # Python program to find the element that appears odd number of times
# O(log n) approach

# Binary search based function
# Returns the element that appears odd number of times
def search(arr, low, high):

    # Base case
    if low > high:
        return None
    if low == high:
        return arr[low]

    # Find the middle point
    mid = (low + high)/2;

    # If mid is even
    if mid%2 == 0:
```

```
        # If the element next to mid is same as mid,
        # then output element lies on right side,
        # else on left side
        if arr[mid] == arr[mid+1]:
            return search(arr, mid+2, high)
        else:
            return search(arr, low, mid)

    else:
        # else if mid is odd

        if arr[mid] == arr[mid-1]:
            return search(arr, mid+1, high)
        else:
            # (mid-1) because target element can only exist at even place
            return search(arr, low, mid-1)


# Test array
arr = [ 1, 1, 2, 2, 1, 1, 2, 2, 13, 1, 1, 40, 40 ]

result = search(arr, 0, len(arr)-1 )

if result is not None:
    print "The required element is %d " % result
else:
    print "Invalid array"
```

C#

```
 // C# program to find the element
// that appears odd number of time
using System;

class GFG  {

    // A Binary Search based function to find
    // the element that appears odd times
    static void search(int []arr, int low, int high)
    {
        // Base cases
        if (low > high)
        return;
        if (low == high)
        {
            Console.WriteLine("The required element is "+
                                            arr[low]);
            return;
```

```
        }

        // Find the middle point
        int mid = (low + high)/2;

        // If mid is even and element next to mid is
        // same as mid, then output element lies on
        // right side, else on left side
        if (mid % 2 == 0)
        {
            if (arr[mid] == arr[mid + 1])
                search(arr, mid + 2, high);
            else
                search(arr, low, mid);
        }

        // If mid is odd
        else
        {
            if (arr[mid] == arr[mid - 1])
                search(arr, mid + 1, high);
            else
                search(arr, low, mid - 1);
        }
    }

    // Driver program
    public static void Main()
    {
        int []arr = {1, 1, 2, 2, 1, 1, 2, 2, 13,
                                    1, 1, 40, 40};
        int len = arr.Length;
        search(arr, 0, len-1);
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to find the
// element that appears odd
// number of times

// A Binary Search based
// function to find the
// element that appears odd times
```

```
function search($arr, $low, $high)
{
    // Base cases
    if ($low > $high)
    return;
    if ($low == $high)
    {
        echo "The required element is ",
                            $arr[$low];
        return;
    }

    // Find the middle point
    $mid = ($low + $high) / 2;

    // If mid is even and element
    // next to mid is same as mid,
    // then output element lies on
    // right side, else on left side
    if ($mid % 2 == 0)
    {
        if ($arr[$mid] == $arr[$mid + 1])
            search($arr, $mid + 2, $high);
        else
            search($arr, $low, $mid);
    }

    // If mid is odd
    else
    {
        if ($arr[$mid] == $arr[$mid - 1])
            search($arr, $mid + 1, $high);
        else
            search($arr, $low, $mid - 1);
    }
}

// Driver Code
$arr = array(1, 1, 2, 2, 1, 1, 2,
             2, 13, 1, 1, 40, 40);
$len = count($arr);;
search($arr, 0, $len - 1);

// This code is contributed by anuj_67.
?>
```

Output :

```
The required element is 13
```

Time Complexity: O(Log n)

This article is contributed by Mehboob Elahi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : Sam007, vt_m

Source

https://www.geeksforgeeks.org/find-the-element-that-odd-number-of-times-in-olog-n-time/

Chapter 101

Find the one missing number in range

Find the one missing number in range - GeeksforGeeks

Given an array of size n. It is also given that range of numbers is from smallestNumber to smallestNumber + n where **smallestNumber** is the smallest number in array. The array contains number in this range but one number is missing so the task is to find this missing number.

Examples:

```
Input  :  arr[] = {13, 12, 11, 15}
Output :  14

Input  :  arr[] = {33, 36, 35, 34};
Output : 37
```

The problem is very close to find missing number.

There are many approaches to solve this problem.
A **simple approach** is to first find minimum, then one by one search all elements. Time complexity of this approach is O(n*n)

A **better solution** is to sort the array. Then traverse the array and find the first element which is not present. Time complexity of this approach is O(n Log n)

The **best solution** is to first XOR all the numbers. Then XOR this result to all numbers from smallest number to n+smallestNumber. The XOR is our result.

Example:-

```
arr[n] = {13, 12, 11, 15}
```

```
smallestNumber = 11

first find the xor of this array
13^12^11^15 = 5

Then find the XOR first number to first number + n
11^12^13^14^15 = 11;

Then xor these two number's
5^11 = 14 // this is the missing number
```

C++

```
 // CPP program to find missing
// number in a range.
#include <bits/stdc++.h>
using namespace std;

// Find the missing number
// in a range
int missingNum(int arr[], int n)
{
    int minvalue = *min_element(arr, arr+n);

    // here we xor of all the number
    int xornum = 0;
    for (int i = 0; i < n; i++) {
        xornum ^= (minvalue) ^ arr[i];
        minvalue++;
    }

    // xor last number
    return xornum ^ minvalue;
}

// Driver code
int main()
{
    int arr[] = { 13, 12, 11, 15 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << missingNum(arr, n);
    return 0;
}
```

Java

```
 // Java program to find
```

```
// missing number in a range.
import java.io.*;
import java.util.*;

class GFG {

// Find the missing number in a range
static int missingNum(int arr[], int n)
{
    List<Integer> list = new ArrayList<>(arr.length);
    for (int i :arr)
    {
        list.add(Integer.valueOf(i));
    }
    int minvalue = Collections.min(list);;

    // here we xor of all the number
    int xornum = 0;
    for (int i = 0; i < n; i++) {
        xornum ^= (minvalue) ^ arr[i];
        minvalue++;
    }

    // xor last number
    return xornum ^ minvalue;
}

public static void main (String[] args) {
     int arr[] = { 13, 12, 11, 15 };
    int n = arr.length;
    System.out.println(missingNum(arr, n));

    }
}

//This code is contributed by Gitanjali.
```

Python3

```
 # python3 program to check
# missingnumber in a range

# Find the missing number
# in a range
def missingNum( arr,  n):

    minvalue = min(arr)
```

```
    # here we xor of all the number
    xornum = 0
    for  i in range (0,n):
        xornum ^= (minvalue) ^ arr[i]
        minvalue = minvalue+1


    # xor last number
    return xornum ^ minvalue

# Driver method
arr = [ 13, 12, 11, 15 ]
n = len(arr)
print (missingNum(arr, n))

# This code is contributed by Gitanjali.
```

Output:

```
14
```

Source

https://www.geeksforgeeks.org/find-one-missing-number-range/

Chapter 102

Find the only missing number in a sorted array

Find the only missing number in a sorted array - GeeksforGeeks

You are given a sorted array of N integers from 1 to N with one number missing find the missing number Expected time complexity **O(logn)**
Examples:

```
Input :ar[] = {1, 3, 4, 5}
Output : 2

Input : ar[] = {1, 2, 3, 4, 5, 7, 8}
Output : 6
```

A **simple solution** is to linearly traverse the given array. Find the point where current element is not one more than previous.

An **efficient solution** is to use binary search. We use the index to search for the missing element and modified binary search. If element at mid != index+1 and this is first missing element then mid + 1 is the missing element. Else if this is not first missing element but ar[mid] != mid+1 search in left half. Else search in right half and if left>right then no element is missing.

C++

```
 // CPP program to find the only missing element.
#include <iostream>
using namespace std;

int findmissing(int ar[], int N)
{
```

```
    int l = 0, r = N - 1;
    while (l <= r) {

        int mid = (l + r) / 2;

        // If this is the first element
        // which is not index + 1, then
        // missing element is mid+1
        if (ar[mid] != mid + 1 &&
                        ar[mid - 1] == mid)
            return mid + 1;

        // if this is not the first missing
        // element search in left side
        if (ar[mid] != mid + 1)
            r = mid - 1;

        // if it follows index+1 property then
        // search in right side
        else
            l = mid + 1;
    }

    // if no element is missing
    return -1;
}

// Driver code
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 7, 8};
    int N = sizeof(arr)/sizeof(arr[0]);
    cout << findmissing(arr, N);
    return 0;
}
```

Java

```
 // Java program to find
// the only missing element.
class GFG
{
static int findmissing(int [] ar, int N)
{

    int l = 0, r = N - 1;
    while (l <= r)
    {
```

```
        int mid = (l + r) / 2;

        // If this is the first element
        // which is not index + 1, then
        // missing element is mid+1
        if (ar[mid] != mid + 1 &&
            ar[mid - 1] == mid)
            return (mid + 1);

        // if this is not the first
        // missing element search
        // in left side
        if (ar[mid] != mid + 1)
            r = mid - 1;

        // if it follows index+1
        // property then search
        // in right side
        else
            l = mid + 1;
    }

// if no element is missing
return -1;
}

// Driver code
public static void main(String [] args)
{
    int arr[] = {1, 2, 3, 4, 5, 7, 8};
    int N = arr.length;
    System.out.println(findmissing(arr, N));
}
}

// This code is contributed
// by Shivi_Aggarwal
```

Python3

```
 # PYTHON 3 program to find
# the only missing element.
def findmissing(ar, N):
    l = 0
    r = N - 1
    while (l <= r):
        mid = (l + r) / 2
        mid= int (mid)
```

```
    # If this is the first element
    # which is not index + 1, then
    # missing element is mid+1
        if(ar[mid] != mid + 1 and
           ar[mid - 1] == mid):
            return (mid + 1)

    # if this is not the first
    # missing element search
    # in left side
        elif(ar[mid] != mid + 1):
            r = mid - 1

    # if it follows index+1
    # property then search
    # in right side
        else:
            l = mid + 1

    # if no element is missing
    return (-1)

def main():
    ar= [1, 2, 3, 4, 5, 7, 8]
    N = len(ar)
    res= findmissing(ar, N)
    print (res)
if __name__ == "__main__":
    main()

# This code is contributed
# by Shivi_Aggarwal
```

C#

```
 // C# program to find
// the only missing element.
using System;

class GFG
{
static int findmissing(int []ar,
                       int N)
{

    int l = 0, r = N - 1;
    while (l <= r)
```

```
    {
        int mid = (l + r) / 2;

        // If this is the first element
        // which is not index + 1, then
        // missing element is mid+1
        if (ar[mid] != mid + 1 &&
            ar[mid - 1] == mid)
            return (mid + 1);

        // if this is not the first
        // missing element search
        // in left side
        if (ar[mid] != mid + 1)
            r = mid - 1;

        // if it follows index+1
        // property then search
        // in right side
        else
            l = mid + 1;
    }

// if no element is missing
return -1;
}

// Driver code
public static void Main()
{
    int []arr = {1, 2, 3, 4, 5, 7, 8};
    int N = arr.Length;
    Console.WriteLine(findmissing(arr, N));
}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

PHP

```
 <?php
// PHP program to find
// the only missing element.
function findmissing(&$ar, $N)
{
    $r = $N - 1;
    $l = 0;
```

```
    while ($l <= $r)
    {
        $mid = ($l + $r) / 2;

        // If this is the first element
        // which is not index + 1, then
        // missing element is mid+1
        if ($ar[$mid] != $mid + 1 &&
            $ar[$mid - 1] == $mid)
            return ($mid + 1);

        // if this is not the first
        // missing element search
        // in left side
        if ($ar[$mid] != $mid + 1)
            $r = $mid - 1;

        // if it follows index+1
        // property then search
        // in right side
        else
            $l = $mid + 1;
    }

    // if no element is missing
    return (-1);
}

// Driver Code
$ar = array(1, 2, 3, 4, 5, 7, 8);
$N = sizeof($ar);
echo(findmissing($ar, $N));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output:

```
6
```

Time Complexity:O(Log n)
Auxiliary Space: O(1)

Improved By : Shivi_Aggarwal, Abby_akku

Source

https://www.geeksforgeeks.org/find-the-only-missing-number-in-a-sorted-array/

Chapter 103

Find the only repetitive element between 1 to n-1

Find the only repetitive element between 1 to n-1 - GeeksforGeeks

We are given an array arr[] of size n. Numbers are from 1 to (n-1) in random order. The array has only one repetitive element. We need to find the repetitive element.

Examples :

```
Input  : a[] = {1, 3, 2, 3, 4}
Output : 3

Input  : a[] = {1, 5, 1, 2, 3, 4}
Output : 1
```

Method 1 (Simple) We use two nested loops. The outer loop traverses through all elements and the inner loop check if the element picked by outer loop appears anywhere else.

Time Complexity : O(n*n)

Method 2 (Using Sum Formula): We know sum of first n-1 natural numbersis (n – 1)*n/2. We compute sum of array elements and subtract natural number sum from it to find the only missing element.

C++

```
 // CPP program to find the only repeating
// element in an array where elements are
// from 1 to n-1.
#include <bits/stdc++.h>
using namespace std;
```

```
int findRepeating(int arr[], int n)
{
   // Find array sum and subtract sum
   // first n-1 natural numbers from it
   // to find the result.
   return accumulate(arr , arr+n , 0) -
                  ((n - 1) * n/2);
}

// driver code
int main()
{
    int arr[] = { 9, 8, 2, 6, 1, 8, 5, 3, 4, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findRepeating(arr, n);
    return 0;
}
```

Output :

```
8
```

Time Complexity : O(n)
Auxiliary Space : O(1)
Causes overflow for large arrays.

Method 3 (Use Hashing): Use a hash table to store elements visited. If a seen element appears again, we return it.

C++

```
 // CPP program to find the only repeating
// element in an array where elements are
// from 1 to n-1.
#include <bits/stdc++.h>
using namespace std;

int findRepeating(int arr[], int n)
{
   unordered_set<int> s;
   for (int i=0; i<n; i++)
   {
      if (s.find(arr[i]) != s.end())
         return arr[i];
      s.insert(arr[i]);
   }
```

```
   // If input is correct, we should
   // never reach here
   return -1;
}

// driver code
int main()
{
    int arr[] = { 9, 8, 2, 6, 1, 8, 5, 3, 4, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findRepeating(arr, n);
    return 0;
}
```

Output :

```
8
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Method 4(Use XOR): The idea is based on the fact that x ^ x = 0 and x ^ y = y ^ x.
1) Compute XOR of elements from 1 to n-1.
2) Compute XOR of array elements.
3) XOR of above two would be our result.

C++

```
 // CPP program to find the only repeating
// element in an array where elements are
// from 1 to n-1.
#include <bits/stdc++.h>
using namespace std;

int findRepeating(int arr[], int n)
{

   // res is going to store value of
   // 1 ^ 2 ^ 3 .. ^ (n-1) ^ arr[0] ^
   // arr[1] ^ .... arr[n-1]
   int res = 0;
   for (int i=0; i<n-1; i++)
       res = res ^ (i+1) ^ arr[i];
   res = res ^ arr[n-1];
```

```
   return res;
}

// driver code
int main()
{
    int arr[] = { 9, 8, 2, 6, 1, 8, 5, 3, 4, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findRepeating(arr, n);
    return 0;
}
```

Java

```
 // Java program to find the only repeating
// element in an array where elements are
// from 1 to n-1.
class GFG
{

    static int findRepeating(int arr[], int n)
    {

        // res is going to store value of
        // 1 ^ 2 ^ 3 .. ^ (n-1) ^ arr[0] ^
        // arr[1] ^ .... arr[n-1]
        int res = 0;
        for (int i = 0; i < n - 1; i++)
            res = res ^ (i + 1) ^ arr[i];
        res = res ^ arr[n - 1];

        return res;
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 9, 8, 2, 6, 1, 8, 5, 3, 4, 7 };
        int n = arr.length;
        System.out.println(findRepeating(arr, n));
    }
}

// This code is contributed by
// Smitha Dinesh Semwal.
```

Python3

```
 # Python3 program to find the only
# repeating element in an array where
# elements are from 1 to n-1.

def findRepeating(arr, n):

    # res is going to store value of
    # 1 ^ 2 ^ 3 .. ^ (n-1) ^ arr[0] ^
    # arr[1] ^ .... arr[n-1]
    res = 0
    for i in range(0, n-1):
        res = res ^ (i+1) ^ arr[i]
    res = res ^ arr[n-1]

    return res

# Driver code
arr = [9, 8, 2, 6, 1, 8, 5, 3, 4, 7]
n = len(arr)
print(findRepeating(arr, n))

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# program to find the
// only repeating element
// in an array where elements
// are from 1 to n-1.
using System;

class GFG
{
    static int findRepeating(int []arr,
                             int n)
    {

        // res is going to store
        // value of 1 ^ 2 ^ 3 ..
        // ^ (n-1) ^ arr[0] ^
        // arr[1] ^ .... arr[n-1]
        int res = 0;
        for (int i = 0; i < n - 1; i++)
            res = res ^ (i + 1) ^ arr[i];
        res = res ^ arr[n - 1];

        return res;
    }
```

```
    // Driver code
    public static void Main()
    {
        int []arr = { 9, 8, 2, 6, 1,
                      8, 5, 3, 4, 7 };
        int n = arr.Length;
        Console.Write(findRepeating(arr, n));
    }
}

// This code is contributed
// by Smitha Dinesh Semwal.
```

PHP

```
 <?php
// PHP program to find the only repeating
// element in an array where elements are
// from 1 to n-1.

function findRepeating($arr, $n)
{

    // res is going to store value of
    // 1 ^ 2 ^ 3 .. ^ (n-1) ^ arr[0] ^
    // arr[1] ^ .... arr[n-1]
    $res = 0;
    for($i = 0; $i < $n - 1; $i++)
        $res = $res ^ ($i + 1) ^ $arr[$i];
    $res = $res ^ $arr[$n - 1];

    return $res;
}

    // Driver Code
    $arr =array(9, 8, 2, 6, 1, 8, 5, 3, 4, 7);
    $n = sizeof($arr) ;
    echo findRepeating($arr, $n);

// This code is contributed by ajit
?>
```

Output:

```
8
```

Time Complexity : O(n)
Auxiliary Space : O(1)

Method 5 : Using indexing.
1. Iterate through the array.
2. For every index visit a[index], if it is positive change the sign of element at a[index] index, else print the element.

C++

```
 // CPP program to find the only
// repeating element in an array
// where elements are from 1 to n-1.
#include <bits/stdc++.h>
using namespace std;

// Function to find repeted element
int findRepeating(int arr[], int n)
{
    int missingElement = 0;

    // indexing based
    for (int i = 0; i < n; i++){

        int element = arr[abs(arr[i])];

        if(element < 0){
            missingElement = arr[i];
            break;
        }

    arr[abs(arr[i])] = -arr[abs(arr[i])];
}

return abs(missingElement);

}

// driver code
int main()
{
    int arr[] = { 5, 4, 3, 9, 8,
                  9, 1, 6, 2, 5};

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << findRepeating(arr, n);
```

```
    return 0;
}
```

Java

```
 // Java program to find the only
// repeating element in an array
// where elements are from 1 to n-1.
import java.lang.Math.*;

class GFG
{
    // Function to find repeted element
    static int findRepeating(int arr[], int n)
    {
        int missingElement = 0;

        // indexing based
        for (int i = 0; i < n; i++){

            int element = arr[Math.abs(arr[i])];

            if(element < 0){
                missingElement = arr[i];
                break;
            }

        arr[Math.abs(arr[i])] = -arr[Math.abs(arr[i])];
    }

    return Math.abs(missingElement);

    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 5, 4, 3, 9, 8,
                      9, 1, 6, 2, 5};

        int n = arr.length;

        System.out.println(findRepeating(arr, n));

    }
}
// This code is contributed by
// Smitha Dinesh Semwal.
```

Python3

```
 # Python3 program to find the only
# repeating element in an array
# where elements are from 1 to n-1.

# Function to find repeted element
def findRepeating(arr, n):

    missingElement = 0

    # indexing based
    for i in range(0, n):

        element = arr[abs(arr[i])]

        if(element < 0):
            missingElement = arr[i]
            break

        arr[abs(arr[i])] = -arr[abs(arr[i])]

    return abs(missingElement)

# Driver code
arr = [5, 4, 3, 9, 8, 9, 1, 6, 2, 5]
n = len(arr)
print(findRepeating(arr, n))

# This code is contributed by Smitha Dinesh Semwal.
```

PHP

```
 <?php
// PHP program to find the only
// repeating element in an array
// where elements are from 1 to n-1.

// Function to find repeted element
function findRepeating($arr, $n)
{
    $missingElement = 0;

    // indexing based
    for ($i = 0; $i < $n; $i++)
    {
```

```
        $element = $arr[abs($arr[$i])];

        if($element < 0)
        {
            $missingElement = $arr[$i];
            break;
        }

    $arr[abs($arr[$i])] = -$arr[abs($arr[$i])];
}

return abs($missingElement);

}

// Driver Code
$arr = array (5, 4, 3, 9, 8,
              9, 1, 6, 2, 5);

$n = sizeof($arr);

echo findRepeating($arr, $n);

// This code is contributed by ajit
?>
```

Output :

```
9
```

Time Complexiy : O(n)
Auxiliary Space : O(1)

Improved By : jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/find-repetitive-element-1-n-1/

Chapter 104

Find the repeating and the missing | Added 3 new methods

Find the repeating and the missing | Added 3 new methods - GeeksforGeeks

Improved By : vt_m

Source

https://www.geeksforgeeks.org/find-a-repeating-and-a-missing-number/

Chapter 105

Find the row with maximum number of 1s

Find the row with maximum number of 1s - GeeksforGeeks

Given a boolean 2D array, where each row is sorted. Find the row with the maximum number of 1s.

```
Example
Input matrix
0 1 1 1
0 0 1 1
1 1 1 1  // this row has maximum 1s
0 0 0 0

Output: 2
```

A simple method is to do a row wise traversal of the matrix, count the number of 1s in each row and compare the count with max. Finally, return the index of row with maximum 1s. The time complexity of this method is O(m*n) where m is number of rows and n is number of columns in matrix.

We can do better. Since each row is sorted, we can **use Binary Search** to count of 1s in each row. We find the index of first instance of 1 in each row. The count of 1s will be equal to total number of columns minus the index of first 1.

See the following code for implementation of the above approach.

C/C++

```
 // CPP program to find the row
// with maximum number of 1s
```

```
#include <stdio.h>
#define R 4
#define C 4

// Function to find the index of first index
// of 1 in a boolean array arr[]
int first(bool arr[], int low, int high)
{
if(high >= low)
{
    // Get the middle index
    int mid = low + (high - low)/2;

    // Check if the element at middle index is first 1
    if ( ( mid == 0 || arr[mid-1] == 0) && arr[mid] == 1)
    return mid;

    // If the element is 0, recur for right side
    else if (arr[mid] == 0)
    return first(arr, (mid + 1), high);

    // If element is not first 1, recur for left side
    else
    return first(arr, low, (mid -1));
}
return -1;
}

// Function that returns index of row
// with maximum number of 1s.
int rowWithMax1s(bool mat[R][C])
{
    // Initialize max values
    int max_row_index = 0, max = -1;

    // Traverse for each row and count number of 1s
    // by finding the index of first 1
    int i, index;
    for (i = 0; i < R; i++)
    {
    index = first (mat[i], 0, C-1);
    if (index != -1 && C-index > max)
    {
        max = C - index;
        max_row_index = i;
    }
    }
```

```
    return max_row_index;
}

// Driver Code
int main()
{
    bool mat[R][C] = { {0, 0, 0, 1},
                       {0, 1, 1, 1},
                       {1, 1, 1, 1},
                       {0, 0, 0, 0}};

    printf("Index of row with maximum 1s is %d "
                                , rowWithMax1s(mat));

    return 0;
}
```

Java

```
 // Java program to find the row
// with maximum number of 1s
import java.io.*;

class GFG {
    static int R = 4, C = 4;
    // Function to find the index of first index
    // of 1 in a boolean array arr[]
    static int first(int arr[], int low, int high)
    {
        if (high >= low) {
            // Get the middle index
            int mid = low + (high - low) / 2;

            // Check if the element at middle index is first 1
            if ((mid == 0 || (arr[mid - 1] == 0)) && arr[mid] == 1)
                return mid;

            // If the element is 0, recur for right side
            else if (arr[mid] == 0)
                return first(arr, (mid + 1), high);

            // If element is not first 1, recur for left side
            else
                return first(arr, low, (mid - 1));
        }
        return -1;
    }
```

```
    // Function that returns index of row
    // with maximum number of 1s.
    static int rowWithMax1s(int mat[][])
    {
        // Initialize max values
        int max_row_index = 0, max = -1;

        // Traverse for each row and count number of
        // 1s by finding the index of first 1
        int i, index;
        for (i = 0; i < R; i++) {
            index = first(mat[i], 0, C - 1);
            if (index != -1 && C - index > max) {
                max = C - index;
                max_row_index = i;
            }
        }

        return max_row_index;
    }
    // Driver Code
    public static void main(String[] args)
    {
        int mat[][] = { { 0, 0, 0, 1 },
                        { 0, 1, 1, 1 },
                        { 1, 1, 1, 1 },
                        { 0, 0, 0, 0 } };
        System.out.println("Index of row with maximum 1s is "
                                            + rowWithMax1s(mat));
    }
}

// This code is contributed by 'Gitanjali'.
```

Python 3

```
 # Python3 program to find the row
# with maximum number of 1s

# Function to find the index
# of first index of 1 in a
# boolean array arr[]
def first( arr, low, high):
    if high >= low:

        # Get the middle index
        mid = low + (high - low)//2
```

```
        # Check if the element at
        # middle index is first 1
        if (mid == 0 or arr[mid - 1] == 0) and arr[mid] == 1:
            return mid

        # If the element is 0,
        # recur for right side
        elif arr[mid] == 0:
            return first(arr, (mid + 1), high)

        # If element is not first 1,
        # recur for left side
        else:
            return first(arr, low, (mid - 1))
    return -1

# Function that returns
# index of row with maximum
# number of 1s.
def rowWithMax1s( mat):

    # Initialize max values
    R = len(mat)
    C = len(mat[0])
    max_row_index = 0
    max = -1

    # Traverse for each row and
    # count number of 1s by finding
    #  the index of first 1
    for i in range(0, R):
        index = first (mat[i], 0, C - 1)
        if index != -1 and C - index > max:
            max = C - index
            max_row_index = i

    return max_row_index

# Driver Code
mat = [[0, 0, 0, 1],
       [0, 1, 1, 1],
       [1, 1, 1, 1],
       [0, 0, 0, 0]]
print ("Index of row with maximum 1s is",
      rowWithMax1s(mat))

# This code is contributed
# by shreyanshi_arun
```

Output:

```
Index of row with maximum 1s is 2
```

Time Complexity: O(mLogn) where m is number of rows and n is number of columns in matrix.

The above solution **can be optimized further**. Instead of doing binary search in every row, we first check whether the row has more 1s than max so far. If the row has more 1s, then only count 1s in the row. Also, to count 1s in a row, we don't do binary search in complete row, we do search in before the index of last max.

Following is an optimized version of the above solution.

```
 // The main function that returns index of row with maximum number of 1s.
int rowWithMax1s(bool mat[R][C])
{
    int i, index;

    // Initialize max using values from first row.
    int max_row_index = 0;
    int max = first(mat[0], 0, C-1);

    // Traverse for each row and count number of 1s by finding the index
    // of first 1
    for (i = 1; i < R; i++)
    {
        // Count 1s in this row only if this row has more 1s than
        // max so far

        // Count 1s in this row only if this row has more 1s than
        // max so far
        if (max != -1 && mat[i][C-max-1] == 1)
        {
            // Note the optimization here also
            index = first (mat[i], 0, C-max);

            if (index != -1 && C-index > max)
            {
                max = C - index;
                max_row_index = i;
            }
        }
        else {
            max = first(mat[i], 0, C - 1);
        }
    }
    return max_row_index;
}
```

The worst case time complexity of the above optimized version is also O(mLogn), the will solution work better on average. Thanks to Naveen Kumar Singh for suggesting the above solution.

The worst case of the above solution occurs for a matrix like following.
0 0 0 … 0 1
0 0 0 ..0 1 1
0 … 0 1 1 1
….0 1 1 1 1

Following method works in O(m+n) time complexity in worst case.

Step1: Get the index of first (or leftmost) 1 in the first row.

Step2: Do following for every row after the first row
…IF the element on left of previous leftmost 1 is 0, ignore this row.
…ELSE Move left until a 0 is found. Update the leftmost index to this index and max_row_index to be the current row.

The time complexity is O(m+n) because we can possibly go as far left as we came ahead in the first step.

Following is C++ implementation of this method.

```
 // The main function that returns index of row with maximum number of 1s.
int rowWithMax1s(bool mat[R][C])
{
    // Initialize first row as row with max 1s
    int max_row_index = 0;

    // The function first() returns index of first 1 in row 0.
    // Use this index to initialize the index of leftmost 1 seen so far
    int j = first(mat[0], 0, C-1);
    if (j == -1) // if 1 is not present in first row
      j = C - 1;

    for (int i = 1; i < R; i++)
    {
        // Move left until a 0 is found
        while (j >= 0 && mat[i][j] == 1)
        {
           j = j-1;  // Update the index of leftmost 1 seen so far
           max_row_index = i;  // Update max_row_index
        }
    }
    return max_row_index;
}
```

Thanks to Tylor, Ankan and Palash for their inputs.

Source

https://www.geeksforgeeks.org/find-the-row-with-maximum-number-1s/

Chapter 106

Find the slope of the given number

Find the slope of the given number - GeeksforGeeks

Find the slope of the given number **num**. Slope of a number is the count of the minima and maxima digits in it. A digit is called a minima if the digit is lesser than the digit before and after it. Similarly a digit is called a maxima if the digit is greater than the digit before and after it.

Examples:

```
Input : 1213321
Output : 2
1213321- The highlighted digit '2' is a maxima and
highlighted digit '1' is a minima.

Input : 273299302236131
Output : 6
```

Source: Barclays Interview Experience (On Campus).

Approach: Traverse the digits of the given number from the 2nd digit up to the 2nd last digit. For each digit check whether the digit is greater or smaller than digits before and after it. Get the count of such digits.

C++

```
 // C++ implementation to find slope of a number
#include <bits/stdc++.h>

using namespace std;
```

```
// function to find slope of a number
int slopeOfNum(string num, int n)
{
    // to store slope of the given
    // number 'num'
    int slope = 0;

    // loop from the 2nd digit up to the 2nd last digit
    // of the given number 'num'
    for (int i = 1; i < n - 1; i++) {

        // if the digit is a maxima
        if (num[i] > num[i - 1] && num[i] > num[i + 1])
            slope++;

        // if the digit is a minima
        else if (num[i] < num[i - 1] && num[i] < num[i + 1])
            slope++;
    }

    // required slope
    return slope;
}

// Driver program to test above
int main()
{
    string num = "1213321";
    int n = num.size();
    cout << "Slpoe = "
         << slopeOfNum(num, n);
    return 0;
}
```

Java

```
 // Java implementation to
// find slope of a number
import java.io.*;

class GFG
{

    // function to find
    // slope of a number
    static int slopeOfNum(String num, int n)
    {
        // to store slope of the
```

```
        // given number 'num'
        int slope = 0;

        // loop from the 2nd digit
        // up to the 2nd last digit
        // of the given number 'num'
        for (int i = 1; i < n - 1; i++)
        {

            // if the digit is a maxima
            if (num.charAt(i) > num.charAt(i - 1) &&
                num.charAt(i) > num.charAt(i + 1))
                slope++;

            // if the digit is a minima
            else if (num.charAt(i) < num.charAt(i - 1) &&
                     num.charAt(i) < num.charAt(i + 1))
                slope++;
        }

        // required slope
        return slope;
    }

    // Driver code
    public static void main (String[] args)
    {
        String num = "1213321";
        int n = num.length();
        System.out.println("Slope = " +
                    slopeOfNum(num, n));
    }
}

// This code is contributed by Mahadev99
```

Output:

```
Slope = 2
```

Time complexity: O(n).

Improved By : Mahadev99

Source

https://www.geeksforgeeks.org/find-the-slope-of-the-given-number/

Chapter 107

Find the smallest and second smallest elements in an array

Find the smallest and second smallest elements in an array - GeeksforGeeks

Write an efficient C program to find smallest and second smallest element in an array.

Example:

```
Input:  arr[] = {12, 13, 1, 10, 34, 1}
Output: The smallest element is 1 and
        second Smallest element is 10
```

A **Simple Solution** is to sort the array in increasing order. The first two elements in sorted array would be two smallest elements. Time complexity of this solution is O(n Log n).

A **Better Solution** is to scan the array twice. In first traversal find the minimum element. Let this element be x. In second traversal, find the smallest element greater than x. Time complexity of this solution is O(n).

The above solution requires two traversals of input array.

An **Efficient Solution** can find the minimum two elements in one traversal. Below is complete algorithm.

Algorithm:

```
1) Initialize both first and second smallest as INT_MAX
   first = second = INT_MAX
2) Loop through all the elements.
   a) If the current element is smaller than first, then update first
       and second.
   b) Else if the current element is smaller than second then update
    second
```

Implementation:

C/C++

```
 // C program to find smallest and second smallest elements
#include <stdio.h>
#include <limits.h> /* For INT_MAX */

void print2Smallest(int arr[], int arr_size)
{
    int i, first, second;

    /* There should be atleast two elements */
    if (arr_size < 2)
    {
        printf(" Invalid Input ");
        return;
    }

    first = second = INT_MAX;
    for (i = 0; i < arr_size ; i ++)
    {
        /* If current element is smaller than first
           then update both first and second */
        if (arr[i] < first)
        {
            second = first;
            first = arr[i];
        }

        /* If arr[i] is in between first and second
           then update second  */
        else if (arr[i] < second && arr[i] != first)
            second = arr[i];
    }
    if (second == INT_MAX)
        printf("There is no second smallest element\n");
    else
        printf("The smallest element is %d and second "
               "Smallest element is %d\n", first, second);
}

/* Driver program to test above function */
int main()
{
    int arr[] = {12, 13, 1, 10, 34, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    print2Smallest(arr, n);
    return 0;
```

```
}
```

Java

```
 // Java program to find smallest and second smallest elements
import java.io.*;

class SecondSmallest
{
    /* Function to print first smallest and second smallest
      elements */
    static void print2Smallest(int arr[])
    {
        int first, second, arr_size = arr.length;

        /* There should be atleast two elements */
        if (arr_size < 2)
        {
            System.out.println(" Invalid Input ");
            return;
        }

        first = second = Integer.MAX_VALUE;
        for (int i = 0; i < arr_size ; i ++)
        {
            /* If current element is smaller than first
              then update both first and second */
            if (arr[i] < first)
            {
                second = first;
                first = arr[i];
            }

            /* If arr[i] is in between first and second
               then update second  */
            else if (arr[i] < second && arr[i] != first)
                second = arr[i];
        }
        if (second == Integer.MAX_VALUE)
            System.out.println("There is no second" +
                               "smallest element");
        else
            System.out.println("The smallest element is " +
                               first + " and second Smallest" +
                               " element is " + second);
    }

    /* Driver program to test above functions */
```

```
    public static void main (String[] args)
    {
        int arr[] = {12, 13, 1, 10, 34, 1};
        print2Smallest(arr);
    }
}
/*This code is contributed by Devesh Agrawal*/
```

Python

```
 # Python program to find smallest and second smallest elements
import sys

def print2Smallest(arr):

    # There should be atleast two elements
    arr_size = len(arr)
    if arr_size < 2:
        print "Invalid Input"
        return

    first = second = sys.maxint
    for i in range(0, arr_size):

        # If current element is smaller than first then
        # update both first and second
        if arr[i] < first:
            second = first
            first = arr[i]

        # If arr[i] is in between first and second then
        # update second
        elif (arr[i] < second and arr[i] != first):
            second = arr[i];

    if (second == sys.maxint):
        print "No second smallest element"
    else:
        print 'The smallest element is',first,'and' \
              ' second smallest element is',second

# Driver function to test above function
arr = [12, 13, 1, 10, 34, 1]
print2Smallest(arr)

# This code is contributed by Devesh Agrawal
```

C#

```
 // C# program to find smallest
// and second smallest elements
using System;

class GFG
{

    /* Function to print first smallest
     and second smallest elements */
    static void print2Smallest(int []arr)
    {
        int first, second, arr_size = arr.Length;

        /* There should be atleast two elements */
        if (arr_size < 2)
        {
            Console.Write(" Invalid Input ");
            return;
        }

        first = second = int.MaxValue;

        for (int i = 0; i < arr_size ; i ++)
        {
            /* If current element is smaller than first
            then update both first and second */
            if (arr[i] < first)
            {
                second = first;
                first = arr[i];
            }

            /* If arr[i] is in between first and second
            then update second */
            else if (arr[i] < second && arr[i] != first)
                second = arr[i];
        }
        if (second == int.MaxValue)
            Console.Write("There is no second" +
                            "smallest element");
        else
            Console.Write("The smallest element is " +
                            first + " and second Smallest" +
                            " element is " + second);
    }

    /* Driver program to test above functions */
    public static void Main()
```

```
    {
        int []arr = {12, 13, 1, 10, 34, 1};
        print2Smallest(arr);
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP  program to find smallest and
// second smallest elements

function print2Smallest($arr, $arr_size)
{
    $INT_MAX = 2147483647;

    /* There should be atleast
       two elements */
    if ($arr_size < 2)
    {
        echo(" Invalid Input ");
        return;
    }

    $first = $second = $INT_MAX;
    for ($i = 0; $i < $arr_size ; $i ++)
    {

        /* If current element is
           smaller than first then
           update both first and
           second */
        if ($arr[$i] < $first)
        {
            $second = $first;
            $first = $arr[$i];
        }

        /* If arr[i] is in between
           first and second then
           update second */
        else if ($arr[$i] < $second &&
                 $arr[$i] != $first)
            $second = $arr[$i];
    }
    if ($second == $INT_MAX)
```

```
        echo("There is no second smallest element\n");
    else
        echo "The smallest element is ",$first
             ," and second Smallest element is "
                                    , $second;
}

// Driver Code
$arr = array(12, 13, 1, 10, 34, 1);
$n = count($arr);
print2Smallest($arr, $n)

// This code is contributed by Smitha
?>
```

Output :

```
The smallest element is 1 and second Smallest element is 10
```

The same approach can be used to find the largest and second largest elements in an array.

Time Complexity: O(n)

Related Article:
Minimum and Second minimum elements using minimum comparisons

Improved By : Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/to-find-smallest-and-second-smallest-element-in-an-array/

Chapter 108

Find the smallest positive number missing from an unsorted array | Set 2

Find the smallest positive number missing from an unsorted array | Set 2 - GeeksforGeeks

Given an unsorted array with both positive and negative elements. Find the smallest positive number missing from the array in O(n) time using constant extra space. It is allowed to modify the original array.

Examples:

```
Input:  {2, 3, 7, 6, 8, -1, -10, 15}
Output: 1

Input:  { 2, 3, -7, 6, 8, 1, -10, 15 }
Output: 4

Input: {1, 1, 0, -1, -2}
Output: 2
```

We have discussed an O(n) time and O(1) extra space solution in previous post. In this post another alternative solution is discussed.
We make the value at index corresponding to given array element equal to array element. For example: consider the array = {2, 3, 7, 6, 8, -1, -10, 15}. To mark presence of element 2 in this array, we make arr[2-1] = 2. In array subscript [2-1], 2 is element to be marked and 1 is subtracted because we are mapping an element value range [1, N] on index value range [0, N-1]. But if we make arr[1] = 2, we will loss data stored at arr[1]. To avoid this, we first store value present at arr[1] and then update it. Next we will mark presence of element previously present at arr[1], i.e. 3. Clearly this lead to some type of random traversal over

the array. Now we have to specify a condition to mark the end of this traversal. There are three conditions that mark the end of this traversal:
1. If element to be marked is negative: No need to mark the presence of this element as we are interested in finding the first missing positive integer. So if a negative element is found, simply end the traversal as no more marking of presence of an element is done.
2. If element to be marked is greater than N : No need to mark the presence of this element because if this element is present then certainly it has taken a place of an element in range [1, N] in array of size N and hence ensuring that our answer lies in the range[1, N]. So simply end the traversal as no more marking of presence of an element is done.
3. If presence of current element is already marked: Suppose element to be marked present is val. If arr[val-1] = val, then we have already marked the presence of this element. So simply end the traversal as no more marking of presence of an element is done.
Also note that it is possible that all the elements of array in the range [1, N] are not marked present in current traversal. To ensure that all the elements in the range are marked present, we check each element of the array lying in this range. If element is not marked, then we start a new traversal beginning from that array element.

After we have marked presence of all array elements lying in the range [1, N], we check which index value ind is not equal to ind+1. If arr[ind] is not equal to ind+1, then ind+1 is the smallest positive missing number. Recall that we are mapping index value range [0, N-1] to element value range [1, N], so 1 is added to ind. If no such ind is found, then all elements in the range [1, N] are present in the array. So the first missing positive number is N+1.

How this solution works in O(n) time?
Observe that each element in range [1, N] is traversed at most twice in worst case. First while performing a traversal started from some other element in the range. Second when checking if a new traversal is required to be initiated from this element to mark the presence of unmarked elements. In worst case each element in the range [1, N] are present in the array and thus all N elements are traversed twice. So total computations are 2*n, and hence the time complexity is O(n).

Below is the implementation of above approach:

C++

```
 /* CPP program to find the smallest
  positive missing number */
#include <bits/stdc++.h>
using namespace std;

// Function to find smallest positive
// missing number.
int findMissingNo(int arr[], int n)
{
    // to store current array element
    int val;

    // to store next array element in
```

```
    // current traversal
    int nextval;

    for (int i = 0; i < n; i++) {

        // if value is negative or greater
        // than array size, then it cannot
        // be marked in array. So move to
        // next element.
        if (arr[i] <= 0 || arr[i] > n)
            continue;

        val = arr[i];

        // traverse the array until we
        // reach at an element which
        // is already marked or which
        // could not be marked.
        while (arr[val - 1] != val) {
            nextval = arr[val - 1];
            arr[val - 1] = val;
            val = nextval;
            if (val <= 0 || val > n)
                break;
        }
    }

    // find first array index which is
    // not marked which is also the
    // smallest positive missing
    // number.
    for (int i = 0; i < n; i++) {
        if (arr[i] != i + 1) {
            return i + 1;
        }
    }

    // if all indices are marked, then
    // smallest missing positive
    // number is array_size + 1.
    return n + 1;
}

// Driver code
int main()
{
    int arr[] = { 2, 3, 7, 6, 8, -1, -10, 15 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);
```

```
    int missing = findMissingNo(arr, arr_size);
    cout << "The smallest positive missing number is "
         << missing;
    return 0;
}
```

Java

```
 /* Java program to find the smallest
positive missing number */
import java.io.*;

class GFG {

    // Function to find smallest positive
    // missing number.
    static int findMissingNo(int []arr, int n)
    {
        // to store current array element
        int val;

        // to store next array element in
        // current traversal
        int nextval;

        for (int i = 0; i < n; i++) {

            // if value is negative or greater
            // than array size, then it cannot
            // be marked in array. So move to
            // next element.
            if (arr[i] <= 0 || arr[i] > n)
                continue;

            val = arr[i];

            // traverse the array until we
            // reach at an element which
            // is already marked or which
            // could not be marked.
            while (arr[val - 1] != val) {
                nextval = arr[val - 1];
                arr[val - 1] = val;
                val = nextval;
                if (val <= 0 || val > n)
                    break;
            }
        }
```

```
        // find first array index which is
        // not marked which is also the
        // smallest positive missing
        // number.
        for (int i = 0; i < n; i++) {
            if (arr[i] != i + 1) {
                return i + 1;
            }
        }

        // if all indices are marked, then
        // smallest missing positive
        // number is array_size + 1.
        return n + 1;
    }

    // Driver code
    public static void main (String[] args)
    {
        int arr[] = { 2, 3, 7, 6, 8, -1, -10, 15 };
        int arr_size = arr.length;

        int missing = findMissingNo(arr, arr_size);

        System.out.println( "The smallest positive"
                + " missing number is " + missing);
    }
}

// This code is contributed by anuj_67.
```

C#

```
 /* C# program to find the smallest
positive missing number */
using System;

class GFG
{
    // Function to find smallest
    // positive missing number.
    static int findMissingNo(int []arr,
                             int n)
    {
        // to store current
        // array element
        int val;
```

```
// to store next array element
// in current traversal
int nextval;

for (int i = 0; i < n; i++)
{

    // if value is negative or greater
    // than array size, then it cannot
    // be marked in array. So move to
    // next element.
    if (arr[i] <= 0 || arr[i] > n)
        continue;

    val = arr[i];

    // traverse the array until we
    // reach at an element which
    // is already marked or which
    // could not be marked.
    while (arr[val - 1] != val)
    {
        nextval = arr[val - 1];
        arr[val - 1] = val;
        val = nextval;
        if (val <= 0 || val > n)
            break;
    }
}

// find first array index which
// is not marked which is also
// the smallest positive missing
// number.
for (int i = 0; i < n; i++)
{
    if (arr[i] != i + 1)
    {
        return i + 1;
    }
}

// if all indices are marked,
// then smallest missing
// positive number is
// array_size + 1.
return n + 1;
```

```
    }

    // Driver code
    public static void Main (String[] args)
    {
        int []arr = {2, 3, 7, 6,
                     8, -1, -10, 15};
        int arr_size = arr.Length;

        int missing = findMissingNo(arr, arr_size);

        Console.Write("The smallest positive" +
                        " missing number is " +
                                    missing);
    }
}

// This code is contributed
// by shiv_bhakt.
```

PHP

```
 <?php
// PHP program to find
// the smallest positive
// missing number

// Function to find smallest
// positive missing number.
function findMissingNo($arr, $n)
{
    // to store current
    // array element
    $val;

    // to store next array element
    // in current traversal
    $nextval;

    for ($i = 0; $i < $n; $i++)
    {

        // if value is negative or
        // greater than array size,
        // then it cannot be marked
        // in array. So move to
        // next element.
        if ($arr[$i] <= 0 ||
```

```
            $arr[$i] > $n)
            continue;

        $val = $arr[$i];

        // traverse the array until
        // we reach at an element
        // which is already marked
        // or which could not be marked.
        while ($arr[$val - 1] != $val)
        {
            $nextval = $arr[$val - 1];
            $arr[$val - 1] = $val;
            $val = $nextval;
            if ($val <= 0 ||
                $val > $n)
                break;
        }
    }

    // find first array index
    // which is not marked
    // which is also the smallest
    // positive missing number.
    for ($i = 0; $i < $n; $i++)
    {
        if ($arr[$i] != $i + 1)
        {
            return $i + 1;
        }
    }

    // if all indices are marked,
    // then smallest missing
    // positive number is
    // array_size + 1.
    return $n + 1;
}

// Driver code
$arr = array(2, 3, 7, 6, 8,
            -1, -10, 15);
$arr_size = sizeof($arr) /
            sizeof($arr[0]);
$missing = findMissingNo($arr,
                         $arr_size);
echo "The smallest positive " .
         "missing number is " ,
```

```
                    $missing;

// This code is contributed
// by shiv_bhakt.
?>
```

Output:

```
The smallest positive missing number is 1
```

Time Complexity: O(n)
Auxiliary Space: O(1)

Improved By : vt_m, shiv_bhakt

Source

https://www.geeksforgeeks.org/find-the-smallest-positive-number-missing-from-an-unsorted-array-set-2/

Chapter 109

Find three closest elements from given three sorted arrays

Find three closest elements from given three sorted arrays - GeeksforGeeks

Given three sorted arrays A[], B[] and C[], find 3 elements i, j and k from A, B and C respectively such that max(abs(A[i] – B[j]), abs(B[j] – C[k]), abs(C[k] – A[i])) is minimized. Here abs() indicates absolute value.

Example :

```
Input: A[] = {1, 4, 10}
       B[] = {2, 15, 20}
       C[] = {10, 12}
Output: 10 15 10
10 from A, 15 from B and 10 from C

Input: A[] = {20, 24, 100}
       B[] = {2, 19, 22, 79, 800}
       C[] = {10, 12, 23, 24, 119}
Output: 24 22 23
24 from A, 22 from B and 23 from C
```

We strongly recommend you to minimize your browser and try this yourself first.

A **Simple Solution** is to run three nested loops to consider all triplets from A, B and C. Compute the value of max(abs(A[i] – B[j]), abs(B[j] – C[k]), abs(C[k] – A[i])) for every triplet and return minimum of all values. Time complexity of this solution is $O(n^3)$

A **Better Solution** is to us Binary Search.
1) Iterate over all elements of A[],
    a) Binary search for element just smaller than or equal to in B[] and C[], and note the

difference.
2) Repeat step 1 for B[] and C[].
3) Return overall minimum.

Time complexity of this solution is O(nLogn)

Efficient Solution Let 'p' be size of A[], 'q' be size of B[] and 'r' be size of C[]

```
1)   Start with i=0, j=0 and k=0 (Three index variables for A,
                                  B and C respectively)

//  p, q and r are sizes of A[], B[] and C[] respectively.
2)   Do following while i < p and j < q and k < r
    a) Find min and maximum of A[i], B[j] and C[k]
    b) Compute diff = max(X, Y, Z) - min(A[i], B[j], C[k]).
    c) If new result is less than current result, change
       it to the new result.
    d) Increment the pointer of the array which contains
       the minimum.
```

Note that we increment the pointer of the array which has the minimum, because our goal is to decrease the difference. Increasing the maximum pointer increases the difference. Increase the second maximum pointer can potentially increase the difference.

C++

```
 // C++ program to find 3 elements such that max(abs(A[i]-B[j]), abs(B[j]-
// C[k]), abs(C[k]-A[i])) is minimized.

#include<bits/stdc++.h>
using namespace std;

void findClosest(int A[], int B[], int C[], int p, int q, int r)
{

    int diff = INT_MAX;  // Initialize min diff

    // Initialize result
    int res_i =0, res_j = 0, res_k = 0;

    // Traverse arrays
    int i=0,j=0,k=0;
    while (i < p && j < q && k < r)
    {
        // Find minimum and maximum of current three elements
        int minimum = min(A[i], min(B[j], C[k]));
        int maximum = max(A[i], max(B[j], C[k]));
```

```
        // Update result if current diff is less than the min
        // diff so far
        if (maximum-minimum < diff)
        {
             res_i = i, res_j = j, res_k = k;
             diff = maximum - minimum;
        }

        // We can't get less than 0 as values are absolute
        if (diff == 0) break;

        // Increment index of array with smallest value
        if (A[i] == minimum) i++;
        else if (B[j] == minimum) j++;
        else k++;
    }

    // Print result
    cout << A[res_i] << " " << B[res_j] << " " << C[res_k];
}

// Driver program
int main()
{
    int A[] = {1, 4, 10};
    int B[] = {2, 15, 20};
    int C[] = {10, 12};

    int p = sizeof A / sizeof A[0];
    int q = sizeof B / sizeof B[0];
    int r = sizeof C / sizeof C[0];

    findClosest(A, B, C, p, q, r);
    return 0;
}
```

Java

```
 // Java program to find 3 elements such
// that max(abs(A[i]-B[j]), abs(B[j]-C[k]),
// abs(C[k]-A[i])) is minimized.
import java.io.*;

class GFG {

    static void findClosest(int A[], int B[], int C[],
                                  int p, int q, int r)
    {
```

```
    int diff = Integer.MAX_VALUE; // Initialize min diff

    // Initialize result
    int res_i =0, res_j = 0, res_k = 0;

    // Traverse arrays
    int i = 0, j = 0, k = 0;
    while (i < p && j < q && k < r)
    {
        // Find minimum and maximum of current three elements
        int minimum = Math.min(A[i],
                      Math.min(B[j], C[k]));
        int maximum = Math.max(A[i],
                      Math.max(B[j], C[k]));

        // Update result if current diff is
        // less than the min diff so far
        if (maximum-minimum < diff)
        {
            res_i = i;
            res_j = j;
            res_k = k;
            diff = maximum - minimum;
        }

        // We can't get less than 0
        // as values are absolute
        if (diff == 0) break;

        // Increment index of array
        // with smallest value
        if (A[i] == minimum) i++;
        else if (B[j] == minimum) j++;
        else k++;
    }

    // Print result
    System.out.println(A[res_i] + " " +
                       B[res_j] + " " + C[res_k]);
}

// Driver code
public static void main (String[] args)
{
    int A[] = {1, 4, 10};
    int B[] = {2, 15, 20};
    int C[] = {10, 12};
```

```
        int p = A.length;
        int q = B.length;
        int r = C.length;

        // Function calling
        findClosest(A, B, C, p, q, r);
    }
}

// This code is contributed by Ajit.
```

Python3

```
 # Python program to find 3 elements such
# that max(abs(A[i]-B[j]), abs(B[j]- C[k]),
# abs(C[k]-A[i])) is minimized.
import sys

def findCloset(A, B, C, p, q, r):

    # Initialize min diff
    diff = sys.maxsize

    res_i = 0
    res_j = 0
    res_k = 0

    # Travesre Array
    i = 0
    j = 0
    k = 0
    while(i < p and j < q and k < r):

        # Find minimum and maximum of
        # current three elements
        minimum = min(A[i], min(B[j], C[k]))
        maximum = max(A[i], max(B[j], C[k]));

        # Update result if current diff is
        # less than the min diff so far
        if maximum-minimum < diff:
            res_i = i
            res_j = j
            res_k = k
            diff = maximum - minimum;

        # We can 't get less than 0 as
        # values are absolute
```

```
        if diff == 0:
            break


        # Increment index of array with
        # smallest value
        if A[i] == minimum:
            i = i+1
        elif B[j] == minimum:
            j = j+1
        else:
            k = k+1

    # Print result
    print(A[res_i], " ", B[res_j], " ", C[res_k])

# Driver Program
A = [1, 4, 10]
B = [2, 15, 20]
C = [10, 12]

p = len(A)
q = len(B)
r = len(C)

findCloset(A,B,C,p,q,r)

# This code is contributed by Shrikant13.
```

C#

```
 // C# program to find 3 elements
// such that max(abs(A[i]-B[j]),
// abs(B[j]-C[k]), abs(C[k]-A[i]))
// is minimized.
using System;

class GFG
{
    static void findClosest(int []A, int []B,
                            int []C, int p,
                            int q, int r)
    {
        // Initialize min diff
        int diff = int.MaxValue;

        // Initialize result
        int res_i = 0,
```

```
        res_j = 0,
        res_k = 0;

    // Traverse arrays
    int i = 0, j = 0, k = 0;
    while (i < p && j < q && k < r)
    {
        // Find minimum and maximum
        // of current three elements
        int minimum = Math.Min(A[i],
                      Math.Min(B[j], C[k]));
        int maximum = Math.Max(A[i],
                      Math.Max(B[j], C[k]));

        // Update result if current
        // diff is less than the min
        // diff so far
        if (maximum - minimum < diff)
        {
            res_i = i;
            res_j = j;
            res_k = k;
            diff = maximum - minimum;
        }

        // We can't get less than 0
        // as values are absolute
        if (diff == 0) break;

        // Increment index of array
        // with smallest value
        if (A[i] == minimum) i++;
        else if (B[j] == minimum) j++;
        else k++;
    }

    // Print result
    Console.WriteLine(A[res_i] + " " +
                      B[res_j] + " " +
                      C[res_k]);
}

// Driver code
public static void Main ()
{
    int []A = {1, 4, 10};
    int []B = {2, 15, 20};
    int []C = {10, 12};
```

```
        int p = A.Length;
        int q = B.Length;
        int r = C.Length;

        // Function calling
        findClosest(A, B, C, p, q, r);
    }
}

// This code is contributed
// by anuj_67.
```

Output:

```
10 15 10
```

Time complexity of this solution is O(p + q + r) where p, q and r are sizes of A[], B[] and C[] respectively.

Thanks to Gaurav Ahirwar for suggesting above solutions.

Improved By : shrikanth13, jit_t, vt_m

Source

https://www.geeksforgeeks.org/find-three-closest-elements-from-given-three-sorted-arrays/

Chapter 110

Find whether an array is subset of another array | Added Method 3

Find whether an array is subset of another array | Added Method 3 - GeeksforGeeks

Given two arrays: arr1[0..m-1] and arr2[0..n-1]. Find whether arr2[] is a subset of arr1[] or not. Both the arrays are not in sorted order. It may be assumed that elements in both array are distinct.

Examples:
Input: arr1[] = {11, 1, 13, 21, 3, 7}, arr2[] = {11, 3, 7, 1}
Output: arr2[] is a subset of arr1[]

Input: arr1[] = {1, 2, 3, 4, 5, 6}, arr2[] = {1, 2, 4}
Output: arr2[] is a subset of arr1[]

Input: arr1[] = {10, 5, 2, 23, 19}, arr2[] = {19, 5, 3}
Output: arr2[] is not a subset of arr1[]

Method 1 (Simple)
Use two loops: The outer loop picks all the elements of arr2[] one by one. The inner loop linearly searches for the element picked by outer loop. If all elements are found then return 1, else return 0.

C++

```
 #include<bits/stdc++.h>

/* Return 1 if arr2[] is a subset of
arr1[] */
bool isSubset(int arr1[], int arr2[],
                        int m, int n)
{
```

```
    int i = 0;
    int j = 0;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            if(arr2[i] == arr1[j])
                break;
        }

        /* If the above inner loop was
        not broken at all then arr2[i]
        is not present in arr1[] */
        if (j == m)
            return 0;
    }

    /* If we reach here then all
    elements of arr2[] are present
    in arr1[] */
    return 1;
}

// Driver code
int main()
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);

    if(isSubset(arr1, arr2, m, n))
        printf("arr2[] is subset of arr1[] ");
    else
        printf("arr2[] is not a subset of arr1[]");

    getchar();
    return 0;
}
```

Java

```
 // Java program to find whether an array
// is subset of another array

class GFG {
```

```
    /* Return true if arr2[] is a subset
    of arr1[] */
    static boolean isSubset(int arr1[],
                int arr2[], int m, int n)
    {
        int i = 0;
        int j = 0;
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < m; j++)
                if(arr2[i] == arr1[j])
                    break;

            /* If the above inner loop
            was not broken at all then
            arr2[i] is not present in
            arr1[] */
            if (j == m)
                return false;
        }

        /* If we reach here then all
        elements of arr2[] are present
        in arr1[] */
        return true;
    }

    // Driver code
    public static void main(String args[])
    {
        int arr1[] = {11, 1, 13, 21, 3, 7};
        int arr2[] = {11, 3, 7, 1};

        int m = arr1.length;
        int n = arr2.length;

        if(isSubset(arr1, arr2, m, n))
            System.out.print("arr2[] is "
                  + "subset of arr1[] ");
        else
            System.out.print("arr2[] is "
             + "not a subset of arr1[]");
    }
}
```

C#

```
 // C# program to find whether an array
```

```
// is subset of another array
using System;

class GFG {

    /* Return true if arr2[] is a
    subset of arr1[] */
    static bool isSubset(int []arr1,
               int []arr2, int m, int n)
    {
        int i = 0;
        int j = 0;
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < m; j++)
                if(arr2[i] == arr1[j])
                    break;

            /* If the above inner loop
            was not broken at all then
            arr2[i] is not present in
            arr1[] */
            if (j == m)
                return false;
        }

        /* If we reach here then all
        elements of arr2[] are present
        in arr1[] */
        return true;
    }

    // Driver function
    public static void Main()
    {
        int []arr1 = {11, 1, 13, 21, 3, 7};
        int []arr2 = {11, 3, 7, 1};

        int m = arr1.Length;
        int n = arr2.Length;

        if(isSubset(arr1, arr2, m, n))
        Console.WriteLine("arr2[] is subset"
                          + " of arr1[] ");
        else
        Console.WriteLine("arr2[] is not a "
                      + "subset of arr1[]");
    }
```

```
}

// This code is contributed by Sam007
```

PHP

```
 <?php
/* Return 1 if arr2[] is a subset of
arr1[] */
function isSubset($arr1, $arr2, $m, $n)
{
    $i = 0;
    $j = 0;
    for ($i = 0; $i < $n; $i++)
    {
        for ($j = 0; $j < $m; $j++)
        {
            if($arr2[$i] == $arr1[$j])
                break;
        }

        /* If the above inner loop was
        not broken at all then arr2[i]
        is not present in arr1[] */
        if ($j == $m)
            return 0;
    }

    /* If we reach here then all
    elements of arr2[] are present
    in arr1[] */
    return 1;
}

// Driver code
    $arr1 = array(11, 1, 13, 21, 3, 7);
    $arr2 = array(11, 3, 7, 1);

    $m = count($arr1);
    $n = count($arr2);

    if(isSubset($arr1, $arr2, $m, $n))
        echo "arr2[] is subset of arr1[] ";
    else
        echo "arr2[] is not a subset of arr1[]";

// This code is contributed by anuj_67.
?>
```

Output:

```
arr2[] is subset of arr1[]
```

Time Complexity: O(m*n)

Method 2 (Use Sorting and Binary Search)

```
1) Sort arr1[] O(mLogm)
2) For each element of arr2[], do binary search for it in sorted arr1[].
         a) If the element is not found then return 0.
3) If all elements are present then return 1.
```

C

```
 #include<stdio.h>

/* Fucntion prototypes */
void quickSort(int *arr, int si, int ei);
int binarySearch(int arr[], int low, int high, int x);

/* Return 1 if arr2[] is a subset of arr1[] */
bool isSubset(int arr1[], int arr2[], int m, int n)
{
    int i = 0;

    quickSort(arr1, 0, m-1);
    for (i=0; i<n; i++)
    {
        if (binarySearch(arr1, 0, m-1, arr2[i]) == -1)
           return 0;
    }

    /* If we reach here then all elements of arr2[]
      are present in arr1[] */
    return 1;
}

/* FOLLOWING FUNCTIONS ARE ONLY FOR SEARCHING AND SORTING PURPOSE */
/* Standard Binary Search function*/
int binarySearch(int arr[], int low, int high, int x)
{
  if(high >= low)
  {
    int mid = (low + high)/2;  /*low + (high - low)/2;*/
```

```
    /* Check if arr[mid] is the first occurrence of x.
        arr[mid] is first occurrence if x is one of the following
        is true:
        (i)  mid == 0 and arr[mid] == x
        (ii) arr[mid-1] < x and arr[mid] == x
     */
    if(( mid == 0 || x > arr[mid-1]) && (arr[mid] == x))
      return mid;
    else if(x > arr[mid])
      return binarySearch(arr, (mid + 1), high, x);
    else
      return binarySearch(arr, low, (mid -1), x);
  }
 return -1;
}

void exchange(int *a, int *b)
{
    int temp;
    temp = *a;
    *a   = *b;
    *b   = temp;
}

int partition(int A[], int si, int ei)
{
    int x = A[ei];
    int i = (si - 1);
    int j;

    for (j = si; j <= ei - 1; j++)
    {
        if(A[j] <= x)
        {
            i++;
            exchange(&A[i], &A[j]);
        }
    }
    exchange (&A[i + 1], &A[ei]);
    return (i + 1);
}

/* Implementation of Quick Sort
A[] --> Array to be sorted
si  --> Starting index
ei  --> Ending index
*/
```

```
void quickSort(int A[], int si, int ei)
{
    int pi;    /* Partitioning index */
    if(si < ei)
    {
        pi = partition(A, si, ei);
        quickSort(A, si, pi - 1);
        quickSort(A, pi + 1, ei);
    }
}

/*Driver program to test above functions */
int main()
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);

    if(isSubset(arr1, arr2, m, n))
      printf("arr2[] is subset of arr1[] ");
    else
      printf("arr2[] is not a subset of arr1[] ");

    return 0;
}
```

Java

```
 class Main
{
    /* Return true if arr2[] is a subset of arr1[] */
    static boolean isSubset(int arr1[], int arr2[], int m, int n)
    {
        int i = 0;

        sort(arr1, 0, m-1);
        for (i=0; i<n; i++)
        {
            if (binarySearch(arr1, 0, m-1, arr2[i]) == -1)
               return false;
        }

        /* If we reach here then all elements of arr2[]
          are present in arr1[] */
        return true;
    }
```

```
/* FOLLOWING FUNCTIONS ARE ONLY FOR SEARCHING AND SORTING PURPOSE */
/* Standard Binary Search function*/
static int binarySearch(int arr[], int low, int high, int x)
{
  if(high >= low)
  {
    int mid = (low + high)/2;  /*low + (high - low)/2;*/

    /* Check if arr[mid] is the first occurrence of x.
        arr[mid] is first occurrence if x is one of the following
        is true:
        (i)  mid == 0 and arr[mid] == x
        (ii) arr[mid-1] < x and arr[mid] == x
     */
    if(( mid == 0 || x > arr[mid-1]) && (arr[mid] == x))
      return mid;
    else if(x > arr[mid])
      return binarySearch(arr, (mid + 1), high, x);
    else
      return binarySearch(arr, low, (mid -1), x);
  }
 return -1;
}

/* This function takes last element as pivot,
   places the pivot element at its correct
   position in sorted array, and places all
   smaller (smaller than pivot) to left of
   pivot and all greater elements to right
   of pivot */
static int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low-1); // index of smaller element
    for (int j=low; j<high; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;

            // swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
```

```
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }


    /* The main function that implements QuickSort()
      arr[] --> Array to be sorted,
      low  --> Starting index,
      high  --> Ending index */
    static void sort(int arr[], int low, int high)
    {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi] is
              now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }

    public static void main(String args[])
    {
        int arr1[] = {11, 1, 13, 21, 3, 7};
        int arr2[] = {11, 3, 7, 1};

        int m = arr1.length;
        int n = arr2.length;

        if(isSubset(arr1, arr2, m, n))
          System.out.print("arr2[] is subset of arr1[] ");
        else
          System.out.print("arr2[] is not a subset of arr1[]");
    }
}
```

Time Complexity: O(mLogm + nLogm). Please note that this will be the complexity if an mLogm algorithm is used for sorting which is not the case in above code. In above code Quick Sort is sued and worst case time complexity of Quick Sort is O(m^2)

Method 3 (Use Sorting and Merging)
1) Sort both arrays: arr1[] and arr2[] O(mLogm + nLogn)
2) Use Merge type of process to see if all elements of sorted arr2[] are present in sorted arr1[].

Thanks to **Parthsarthi** for suggesting this method.

C++

```
 #include <bits/stdc++.h>
using namespace std;

/* Return 1 if arr2[] is a subset of arr1[] */
bool isSubset(int arr1[], int arr2[], int m, int n)
{
    int i = 0, j = 0;

    if (m < n)
       return 0;

    sort(arr1, arr1 + m);
    sort(arr2, arr2 + n);
    while (i < n && j < m )
    {
        if( arr1[j] <arr2[i] )
            j++;
        else if( arr1[j] == arr2[i] )
        {
            j++;
            i++;
        }
        else if( arr1[j] > arr2[i] )
            return 0;
    }

    return  (i < n)? false : true;
}

/*Driver program to test above functions */
int main()
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);

    if(isSubset(arr1, arr2, m, n))
```

```
      printf("arr2[] is subset of arr1[] ");
    else
      printf("arr2[] is not a subset of arr1[] ");

    return 0;
}
```

Java

```
 // Java code to find whether an array is subset of
// another array
import java.util.Arrays;
class GFG
{
    /* Return true if arr2[] is a subset of arr1[] */
    static boolean isSubset(int arr1[], int arr2[], int m,
                                                    int n)
    {
        int i = 0, j = 0;

        if(m < n)
        return false;

        Arrays.sort(arr1); //sorts arr1
        Arrays.sort(arr2); // sorts arr2

        while( i < n && j < m )
        {
            if( arr1[j] < arr2[i] )
                j++;
            else if( arr1[j] == arr2[i] )
            {
                j++;
                i++;
            }
            else if( arr1[j] > arr2[i] )
                return false;
        }

        if( i < n )
            return false;
        else
            return true;
    }

    public static void main(String[] args)
    {
        int arr1[] = {11, 1, 13, 21, 3, 7};
```

```
        int arr2[] = {11, 3, 7, 1};

        int m = arr1.length;
        int n = arr2.length;

        if(isSubset(arr1, arr2, m, n))
        System.out.println("arr2 is a subset of arr1");
        else
        System.out.println("arr2 is not a subset of arr1");
    }
}
// This code is contributed by Kamal Rawal
```

C#

```
 // C# code to find whether an array
// is subset of another array
using System;
class GFG {

    // Return true if arr2[] is
    // a subset of arr1[] */
    static bool isSubset(int []arr1,
                         int []arr2,
                         int m,
                         int n)
    {
        int i = 0, j = 0;

        if(m < n)
            return false;

        //sorts arr1
        Array.Sort(arr1);

        // sorts arr2
        Array.Sort(arr2);

        while( i < n && j < m )
        {
            if( arr1[j] < arr2[i] )
                j++;
            else if( arr1[j] == arr2[i] )
            {
                j++;
                i++;
            }
            else if( arr1[j] > arr2[i] )
```

```
                return false;
        }

        if( i < n )
            return false;
        else
            return true;
    }

    // Driver Code
    public static void Main()
    {
        int []arr1 = {11, 1, 13, 21, 3, 7};
        int []arr2 = {11, 3, 7, 1};

        int m = arr1.Length;
        int n = arr2.Length;

        if(isSubset(arr1, arr2, m, n))
            Console.Write("arr2 is a subset of arr1");
        else
            Console.Write("arr2 is not a subset of arr1");
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php

/* Return 1 if arr2[] is a subset of arr1[] */
function isSubset( $arr1, $arr2, $m, $n)
{
    $i = 0; $j = 0;

    if ($m < $n)
        return 0;

    sort($arr1);
    sort($arr2);

    while ($i < $n and $j < $m )
    {
        if( $arr1[$j] <$arr2[$i] )
            $j++;
        else if( $arr1[$j] == $arr2[$i] )
        {
```

```
            $j++;
            $i++;
        }
        else if( $arr1[$j] > $arr2[$i] )
            return 0;
    }

    return ($i < $n) ? false : true;
}

/*Driver program to test above functions */

    $arr1 = array(11, 1, 13, 21, 3, 7);
    $arr2 = array(11, 3, 7, 1);

    $m = count($arr1);
    $n = count($arr2);

    if(isSubset($arr1, $arr2, $m, $n))
        echo "arr2[] is subset of arr1[] ";
    else
        echo "arr2[] is not a subset of arr1[] ";

// This code is contributed by anuj_67.
?>
```

Output:

```
arr2 is a subset of arr1
```

Time Complexity: O(mLogm + nLogn) which is better than method 2. Please note that this will be the complexity if an nLogn algorithm is used for sorting both arrays which is not the case in above code. In above code Quick Sort is sued and worst case time complexity of Quick Sort is O(n^2)

Method 4 (Use Hashing)
1) Create a Hash Table for all the elements of arr1[].
2) Traverse arr2[] and search for each element of arr2[] in the Hash Table. If element is not found then return 0.
3) If all elements are found then return 1.

```
 // Java code to find whether an array is subset of
// another array
import java.util.HashSet;
class GFG
{
    /* Return true if arr2[] is a subset of arr1[] */
```

```
    static boolean isSubset(int arr1[], int arr2[], int m,
                                                    int n)
    {
        HashSet<Integer> hset= new HashSet<>();

        // hset stores all the values of arr1
        for(int i = 0; i < m; i++)
        {
            if(!hset.contains(arr1[i]))
                hset.add(arr1[i]);
        }

        // loop to check if all elements of arr2 also
        // lies in arr1
        for(int i = 0; i < n; i++)
        {
            if(!hset.contains(arr2[i]))
                return false;
        }
        return true;
    }

    public static void main(String[] args)
    {
        int arr1[] = {11, 1, 13, 21, 3, 7};
        int arr2[] = {11, 3, 7, 1};

        int m = arr1.length;
        int n = arr2.length;

        if(isSubset(arr1, arr2, m, n))
        System.out.println("arr2 is a subset of arr1");
        else
        System.out.println("arr2 is not a subset of arr1");
    }
}
// This code is contributed by Kamal Rawal
```

Note that method 1, method 2 and method 4 don't handle the cases when we have duplicates in arr2[]. For example, {1, 4, 4, 2} is not a subset of {1, 4, 2}, but these methods will print it as a subset.

Improved By : Sam007, nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/find-whether-an-array-is-subset-of-another-array-set-1/

Chapter 111

First common element in two linked lists

First common element in two linked lists - GeeksforGeeks

Given two Linked Lists, find the first common element between given linked list i.e., we need to find first node of first list which is also present in second list.
Examples:

```
Input :
   List1: 10->15->4->20
   Lsit2:  8->4->2->10
Output : 10

Input :
   List1: 1->2->3->4
   Lsit2:  5->6->3->8
Output : 3
```

We traverse first list and for every node, we search it in second list. As soon as we find an element in second list, we return it.

```
 // C++ program to find first common element in
// two unsorted linked list
#include <bits/stdc++.h>
using namespace std;

/* Link list node */
struct Node {
    int data;
    struct Node* next;
```

```
};

/* A utility function to insert a node at the
   beginning of a linked list*/
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node =
          (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

/* Returns the first repeating element in linked list*/
int firstCommon(struct Node* head1, struct Node* head2)
{
    // Traverse through every node of first list
    for (; head1 != NULL; head1=head1->next)

       // If current node is present in second list
       for (Node *p = head2; p != NULL; p = p->next)
            if (p->data == head1->data)
                return head1->data;

    // If no common node
    return 0;
}

// Driver code
int main()
{
    struct Node* head1 = NULL;
    push(&head1, 20);
    push(&head1, 5);
    push(&head1, 15);
    push(&head1, 10);

    struct Node* head2 = NULL;
    push(&head2, 10);
    push(&head2, 2);
    push(&head2, 15);
    push(&head2, 8);

    cout << firstCommon(head1, head2);
    return 0;
}
```

Output:

```
10
```

Source

https://www.geeksforgeeks.org/first-common-element-two-linked-lists/

Chapter 112

First strictly greater element in a sorted array in Java

First strictly greater element in a sorted array in Java - GeeksforGeeks

Given a sorted array and a target value, find the first element that is strictly greater than given element.

Examples:

```
Input : arr[] = {1, 2, 3, 5, 8, 12}
        Target = 5
Output : 4 (Index of 8)

Input : {1, 2, 3, 5, 8, 12}
        Target = 8
Output : 5 (Index of 12)

Input : {1, 2, 3, 5, 8, 12}
        Target = 15
Output : -1
```

A **simple solution** is to linearly traverse given array and find first element that is strictly greater. If no such element exists, then return -1.

An **efficient solution** is to use Binary Search. In a general binary search, we are looking for a value which appears in the array. Sometimes, however, we need to find the first element which is either greater than a target.

To see that this algorithm is correct, consider each comparison being made. If we find an element that's no greater than the target element, then it and everything below it can't possibly match, so there's no need to search that region. We can thus search the right half. If we find an element that is larger than the element in question, then anything after it must

also be larger, so they can't be the first element that's bigger and so we don't need to search them. The middle element is thus the last possible place it could be.

Note that on each iteration we drop off at least half the remaining elements from consideration. If the top branch executes, then the elements in the range [low, (low + high) / 2] are all discarded, causing us to lose floor((low + high) / 2) – low + 1 >= (low + high) / 2 – low = (high – low) / 2 elements.

If the bottom branch executes, then the elements in the range [(low + high) / 2 + 1, high] are all discarded. This loses us high – floor(low + high) / 2 + 1 >= high – (low + high) / 2 = (high – low) / 2 elements.

Consequently, we'll end up finding the first element greater than the target in O(lg n) iterations of this process.

```
 // Java program to find first element that
// is strictly greater than given target.

class GfG {
    private static int next(int[] arr, int target)
    {
        int start = 0, end = arr.length - 1;

        int ans = -1;
        while (start <= end) {
            int mid = (start + end) / 2;

            // Move to right side if target is
            // greater.
            if (arr[mid] <= target) {
                start = mid + 1;
            }

            // Move left side.
            else {
                ans = mid;
                end = mid - 1;
            }
        }
        return ans;
    }

    // Driver code
    public static void main(String[] args)
    {
        int[] arr = { 1, 2, 3, 5, 8, 12 };
        System.out.println(next(arr, 8));
    }
}
```

Output:

```
5
```

Source

https://www.geeksforgeeks.org/first-strictly-greater-element-in-a-sorted-array-in-java/

Chapter 113

First strictly smaller element in a sorted array in Java

First strictly smaller element in a sorted array in Java - GeeksforGeeks

Given a sorted array and a target value, find the first element that is strictly smaller than given element.

Examples:

```
Input : arr[] = {1, 2, 3, 5, 8, 12}
        Target = 5
Output : 2 (Index of 3)

Input : {1, 2, 3, 5, 8, 12}
        Target = 8
Output : 3 (Index of 5)

Input : {1, 2, 3, 5, 8, 12}
        Target = 15
Output : -1
```

A **simple solution** is to linearly traverse given array and find first element that is strictly greater. If no such element exists, then return -1.

An **efficient solution** is to useBinary Search. In a general binary search, we are looking for a value which appears in the array. Sometimes, however, we need to find the first element which is either greater than a target.

To see that this algorithm is correct, consider each comparison being made. If we find an element that's greater than the target element, then it and everything above it can't possibly match, so there's no need to search that region. We can thus search the left half. If we find an element that is smaller than the element in question, then anything before it must also

be larger, so they can't be the first element that's smaller and so we don't need to search them. The middle element is thus the last possible place it could be.

Note that on each iteration we drop off at least half the remaining elements from consideration. If the top branch executes, then the elements in the range [low, (low + high) / 2] are all discarded, causing us to lose floor((low + high) / 2) – low + 1 >= (low + high) / 2 – low = (high – low) / 2 elements.

If the bottom branch executes, then the elements in the range [(low + high) / 2 + 1, high] are all discarded. This loses us high – floor(low + high) / 2 + 1 >= high – (low + high) / 2 = (high – low) / 2 elements.

Consequently, we'll end up finding the first element smaller than the target in O(lg n) iterations of this process.

```
 // Java program to find first element that
// is strictly greater than given target.

class GfG {

    private static int next(int[] arr, int target)
    {
        int start = 0, end = arr.length-1;

        int ans = -1;
        while (start <= end) {
            int mid = (start + end) / 2;

            // Move to the left side if the target is smaller
            if (arr[mid] >= target) {
                end = mid - 1;
            }

            // Move right side
            else {
                ans = mid;
                start = mid + 1;
            }
        }
        return ans;
    }

    // Driver code
    public static void main(String[] args)
    {
        int[] arr = { 1, 2, 3, 5, 8, 12 };
        System.out.println(next(arr, 5));
    }
}
```

Output:

```
2
```

Source

https://www.geeksforgeeks.org/first-strictly-smaller-element-in-a-sorted-array-in-java/

Chapter 114

Floor in Binary Search Tree (BST)

Floor in Binary Search Tree (BST) - GeeksforGeeks

Given a Binary Search Tree and a number x, find floor of x in the given BST.

```
Input : x = 14 and root of below tree
            10
           /  \
          5    15
              /  \
             12    30
Output : 12

Input : x = 15 and root of below tree
            10
           /  \
          5    15
              /  \
             12    30
Output : 15
```

A **simple solution** is to traverse the tree using (Inorder or Preorder or Postorder) and keep track of closest smaller or same element. Time complexity of this solution is **O(n)** where n is total number of Nodes in BST.

We can **efficiently** find closes in **O(h)** time where h is height of BST. Algorithm to find the floor of a key in a binary search tree (BST):

```
1 Start at the root Node.
2 If root->data == key,
```

```
     floor of the key is equal
     to the root.
3 Else if root->data > key, then
     floor of the key must lie in the
     left subtree.
4 Else floor may lie in the right subtree
  but only if there is a value lesser than
  or equal to the key.If not, then root is
  the key.
```

For finding ciel of BST you can refer to this article.

```
 // CPP code to find floor of a key in BST
#include <bits/stdc++.h>
using namespace std;

/*Structure of each Node in the tree*/
struct Node {
    int data;
    Node* left, * right;
};

/*This function is used to create and
initialises new Nodes*/
Node* newNode(int key)
{
    Node* temp = new Node;
    temp->left = temp->right =  NULL;
    temp->data = key;
    return temp;
}

/* This function is used to insert
 new values in BST*/
Node* insert(Node* root, int key)
{
    if (!root)
        return newNode(key);
    if (key < root->data)
        root->left = insert(root->left, key);
    else
        root->right = insert(root->right, key);
    return root;
}

/*This function is used to find floor of a key*/
int floor(Node* root, int key)
{
```

```
    if (!root)
        return INT_MAX;

    /* If root->data is equal to key */
    if (root->data == key)
        return root->data;

    /* If root->data is greater than the key */
    if (root->data > key)
        return floor(root->left, key);

    /* Else, the floor may lie in right subtree
      or may be equal to the root*/
    int floorValue = floor(root->right, key);
    return (floorValue <= key) ? floorValue : root->data;
}

int main()
{
    /* Let us create following BST
              7
            /    \
           5      10
         /  \    /  \
        3    6   8   12 */
    Node* root = NULL;
    root = insert(root, 7);
    insert(root, 10);
    insert(root, 5);
    insert(root, 3);
    insert(root, 6);
    insert(root, 8);
    insert(root, 12);
    cout << floor(root, 9) << endl;
    return 0;
}
```

Output:

```
8
```

Source

https://www.geeksforgeeks.org/floor-in-binary-search-tree-bst/

Chapter 115

Floor in a Sorted Array

Floor in a Sorted Array - GeeksforGeeks

Given a sorted array and a value x, the floor of x is the largest element in array smaller than or equal to x. Write efficient functions to find floor of x.

Examples:

```
Input : arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 5
Output : 2
2 is the largest element in arr[] smaller than 5.

Input : arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 20
Output : 19
19 is the largest element in arr[] smaller than 20.

Input : arr[] = {1, 2, 8, 10, 10, 12, 19}, x = 0
Output : -1
Since floor doesn't exist, output is -1.
```

Method 1 (Simple)
A simple solution is linearly traverse input sorted array and search for the first element greater than x. The element just before the found element is floor of x.

C++

```
 // C/C++ program to find floor of a given number
// in a sorted array
#include<stdio.h>

/* An inefficient function to get index of floor
of x in arr[0..n-1] */
```

```
int floorSearch(int arr[], int n, int x)
{
    // If last element is smaller than x
    if (x >= arr[n-1])
        return n-1;

    // If first element is greater than x
    if (x < arr[0])
        return -1;

    // Linearly search for the first element
    // greater than x
    for (int i=1; i<n; i++)
    if (arr[i] > x)
        return (i-1);

    return -1;
}

/* Driver program to check above functions */
int main()
{
    int arr[] = {1, 2, 4, 6, 10, 12, 14};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 7;
    int index = floorSearch(arr, n-1, x);
    if (index == -1)
        printf("Floor of %d doesn't exist in array ", x);
    else
        printf("Floor of %d is %d", x, arr[index]);
    return 0;
}
```

Python3

```
 # Python3 program to find floor of a
# given number in a sorted array

# Function to get index of floor
# of x in arr[low..high]
def floorSearch(arr, low, high, x):

    # If low and high cross each other
    if (low > high):
        return -1

    # If last element is smaller than x
    if (x >= arr[high]):
```

```
        return high

    # Find the middle point
    mid = int((low + high) / 2)

    # If middle point is floor.
    if (arr[mid] == x):
        return mid

    # If x lies between mid-1 and mid
    if (mid > 0 and arr[mid-1] <= x
                and x < arr[mid]):
        return mid - 1

    # If x is smaller than mid,
    # floor must be in left half.
    if (x < arr[mid]):
        return floorSearch(arr, low, mid-1, x)

    # If mid-1 is not floor and x is greater than
    # arr[mid],
    return floorSearch(arr, mid+1, high, x)


# Driver Code
arr = [1, 2, 4, 6, 10, 12, 14]
n = len(arr)
x = 7
index = floorSearch(arr, 0, n-1, x)

if (index == -1):
    print("Floor of", x, "doesn't exist \
                    in array ", end = "")
else:
    print("Floor of", x, "is", arr[index])

# This code is contributed by Smitha Dinesh Semwal.
```

Output:

```
Floor of 7 is 6.
```

Time Complexity : O(n)

Method 2 (Efficient)

The idea is to use Binary Search.

C++

```
 // A C/C++ program to find floor of a given number
// in a sorted array
#include<stdio.h>

/* Function to get index of floor of x in
   arr[low..high] */
int floorSearch(int arr[], int low, int high, int x)
{
    // If low and high cross each other
    if (low > high)
        return -1;

    // If last element is smaller than x
    if (x >= arr[high])
        return high;

    // Find the middle point
    int mid = (low+high)/2;

    // If middle point is floor.
    if (arr[mid] == x)
        return mid;

    // If x lies between mid-1 and mid
    if (mid > 0 && arr[mid-1] <= x && x < arr[mid])
        return mid-1;

    // If x is smaller than mid, floor must be in
    // left half.
    if (x < arr[mid])
        return floorSearch(arr, low, mid-1, x);

    // If mid-1 is not floor and x is greater than
    // arr[mid],
    return floorSearch(arr, mid+1, high, x);
}

/* Driver program to check above functions */
int main()
{
    int arr[] = {1, 2, 4, 6, 10, 12, 14};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 7;
    int index = floorSearch(arr, 0, n-1, x);
```

```
    if (index == -1)
        printf("Floor of %d doesn't exist in array ", x);
    else
        printf("Floor of %d is %d", x, arr[index]);
    return 0;
}
```

Java

```
 // Java program to find floor of
// a given number in a sorted array
import java.io.*;

class GFG {

    /* Function to get index of floor of x in
    arr[low..high] */
    static int floorSearch(int arr[], int low,
                           int high, int x)
    {
        // If low and high cross each other
        if (low > high)
            return -1;

        // If last element is smaller than x
        if (x >= arr[high])
            return high;

        // Find the middle point
        int mid = (low+high)/2;

        // If middle point is floor.
        if (arr[mid] == x)
            return mid;

        // If x lies between mid-1 and mid
        if (mid > 0 && arr[mid-1] <= x && x <
                                    arr[mid])
            return mid-1;

        // If x is smaller than mid, floor
        // must be in left half.
        if (x < arr[mid])
            return floorSearch(arr, low,
                               mid - 1, x);

        // If mid-1 is not floor and x is
        // greater than arr[mid],
```

```
        return floorSearch(arr, mid + 1, high,
                                           x);
    }

    /* Driver program to check above functions */
    public static void main(String[] args)
    {
        int arr[] = {1, 2, 4, 6, 10, 12, 14};
        int n = arr.length;
        int x = 7;
        int index = floorSearch(arr, 0, n - 1,
                                            x);
        if (index == -1)
            System.out.println("Floor of " + x +
                     " dosen't exist in array ");
        else
            System.out.println("Floor of " + x +
                            " is " + arr[index]);
    }
}
// This code is contributed by Prerna Saini
```

Python3

```
 # Python3 program to find floor of a
# given number in a sorted array

# Function to get index of floor
# of x in arr[low..high]
def floorSearch(arr, low, high, x):

    # If low and high cross each other
    if (low > high):
        return -1

    # If last element is smaller than x
    if (x >= arr[high]):
        return high

    # Find the middle point
    mid = int((low + high) / 2)

    # If middle point is floor.
    if (arr[mid] == x):
        return mid

    # If x lies between mid-1 and mid
    if (mid > 0 and arr[mid-1] <= x
```

```
                and x < arr[mid]):
            return mid - 1

        # If x is smaller than mid,
        # floor must be in left half.
        if (x < arr[mid]):
            return floorSearch(arr, low, mid-1, x)

        # If mid-1 is not floor and x is greater than
        # arr[mid],
        return floorSearch(arr, mid+1, high, x)


# Driver Code
arr = [1, 2, 4, 6, 10, 12, 14]
n = len(arr)
x = 7
index = floorSearch(arr, 0, n-1, x)

if (index == -1):
    print("Floor of", x, "doesn't exist\
                    in array ", end = "")
else:
    print("Floor of", x, "is", arr[index])

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# program to find floor of
// a given number in a sorted array
using System;

class GFG {

    /* Function to get index of floor of x in
    arr[low..high] */
    static int floorSearch(int []arr, int low,
                               int high, int x)
    {

        // If low and high cross each other
        if (low > high)
            return -1;

        // If last element is smaller than x
        if (x >= arr[high])
            return high;
```

```
        // Find the middle point
        int mid = (low + high) / 2;

        // If middle point is floor.
        if (arr[mid] == x)
            return mid;

        // If x lies between mid-1 and mid
        if (mid > 0 && arr[mid-1] <= x && x <
                                    arr[mid])
            return mid - 1;

        // If x is smaller than mid, floor
        // must be in left half.
        if (x < arr[mid])
            return floorSearch(arr, low,
                               mid - 1, x);

        // If mid-1 is not floor and x is
        // greater than arr[mid],
        return floorSearch(arr, mid + 1, high,
                                         x);
    }

    /* Driver program to check above functions */
    public static void Main()
    {
        int []arr = {1, 2, 4, 6, 10, 12, 14};
        int n = arr.Length;
        int x = 7;
        int index = floorSearch(arr, 0, n - 1,
                                          x);
        if (index == -1)
            Console.Write("Floor of " + x +
                     " dosen't exist in array ");
        else
            Console.Write("Floor of " + x +
                           " is " + arr[index]);
    }
}

// This code is contributed by nitin mittal.
```

Output:

```
Floor of 7 is 6.
```

Time Complexity : O(Log n)

Improved By : nitin mittal, jit_t

Source

https://www.geeksforgeeks.org/floor-in-a-sorted-array/

Chapter 116

For each element in 1st array count elements less than or equal to it in 2nd array

For each element in 1st array count elements less than or equal to it in 2nd array - GeeksforGeeks

Given two unsorted arrays arr1[] and arr2[]. They may contain duplicates. For each element in arr1[] count elements less than or equal to it in array arr2[].

Source: Amazon Interview Experience | Set 354 (For SDE-2)

Examples:

```
Input : arr1[] = [1, 2, 3, 4, 7, 9]
        arr2[] = [0, 1, 2, 1, 1, 4]
Output : [4, 5, 5, 6, 6, 6]

Input : arr1[] = [5, 10, 2, 6, 1, 8, 6, 12]
        arr2[] = [6, 5, 11, 4, 2, 3, 7]
Output : [4, 6, 1, 5, 0, 6, 5, 7]
```

Naive Approach: Using two loops, outer loop for elements of array **arr1**[] and inner loop for elements of array **arr2**[]. Then for each element of **arr1**[], count elements less than or equal to it in **arr2**[].
Time complexity: O(m * n), considering **arr1**[] and **arr2**[] are of sizes **m** and **n** respectively.

Efficient Approach: Sort the elements of 2nd array, i.e., array **arr2**[]. Then perform a modified binary search on array **arr2**[]. For each element **x** of array **arr1**[], find the last index of the largest element smaller than or equal to **x** in sorted array **arr2**[].

C++

```
 // C++ implementation of For each element in 1st
// array count elements less than or equal to it
// in 2nd array
#include <bits/stdc++.h>

using namespace std;

// function returns the index of largest element
// smaller than equal to 'x' in 'arr'. For duplicates
// it returns the last index of occurrence of required
// element. If no such element exits then it returns -1.
int binary_search(int arr[], int l, int h, int x)
{
    while (l <= h)
    {
        int mid = (l+h) / 2;

        // if 'x' is greater than or equal to arr[mid],
        // then search in arr[mid+1...h]
        if (arr[mid] <= x)
            l = mid + 1;

        // else search in arr[l...mid-1]
        else
            h = mid - 1;
    }

    // required index
    return h;
}

// function to count for each element in 1st array,
// elements less than or equal to it in 2nd array
void countEleLessThanOrEqual(int arr1[], int arr2[],
                             int m, int n)
{
    // sort the 2nd array
    sort(arr2, arr2+n);

    // for each element of 1st array
    for (int i=0; i<m; i++)
    {
        // last index of largest element
        // smaller than or equal to x
        int index = binary_search(arr2, 0, n-1, arr1[i]);

        // required count for the element arr1[i]
        cout << (index+1) << " ";
```

```
    }
}

// Driver program to test above
int main()
{
    int arr1[] = {1, 2, 3, 4, 7, 9};
    int arr2[] = {0, 1, 2, 1, 1, 4};
    int m = sizeof(arr1) / sizeof(arr1[0]);
    int n = sizeof(arr2) / sizeof(arr2[0]);
    countEleLessThanOrEqual(arr1, arr2, m, n);
    return 0;
}
```

Java

```
 // Java implementation of For each element in 1st
// array count elements less than or equal to it
// in 2nd array

import java.util.Arrays;

class GFG
{
    // method returns the index of largest element
    // smaller than equal to 'x' in 'arr'. For duplicates
    // it returns the last index of occurrence of required
    // element. If no such element exits then it returns -1.
    static int binary_search(int arr[], int l, int h, int x)
    {
        while (l <= h)
        {
            int mid = (l+h) / 2;

            // if 'x' is greater than or equal to arr[mid],
            // then search in arr[mid+1...h]
            if (arr[mid] <= x)
                l = mid + 1;

            // else search in arr[l...mid-1]
            else
                h = mid - 1;
        }

        // required index
        return h;
    }
```

```
    // method to count for each element in 1st array,
    // elements less than or equal to it in 2nd array
    static void countEleLessThanOrEqual(int arr1[], int arr2[],
                                int m, int n)
    {
        // sort the 2nd array
        Arrays.sort(arr2);

        // for each element of 1st array
        for (int i=0; i<m; i++)
        {
            // last index of largest element
            // smaller than or equal to x
            int index = binary_search(arr2, 0, n-1, arr1[i]);

            // required count for the element arr1[i]
            System.out.print((index+1) + " ");
        }
    }

    // Driver method
    public static void main(String[] args)
    {
        int arr1[] = {1, 2, 3, 4, 7, 9};
        int arr2[] = {0, 1, 2, 1, 1, 4};

        countEleLessThanOrEqual(arr1, arr2, arr1.length, arr2.length);
    }
}
```

Python

```
 # python implementation of For each element in 1st
# array count elements less than or equal to it
# in 2nd array

# function returns the index of largest element
# smaller than equal to 'x' in 'arr'. For duplicates
# it returns the last index of occurrence of required
# element. If no such element exits then it returns -1
def bin_search(arr, n, x):

l = 0
h = n - 1
while(l <= h):
    mid = int((l + h) / 2)
    # if 'x' is greater than or equal to arr[mid],
    # then search in arr[mid + 1...h]
```

```
    if(arr[mid] <= x):
    l = mid + 1;
    else:
    # else search in arr[l...mid-1]
    h = mid - 1
# required index
return h

# function to count for each element in 1st array,
# elements less than or equal to it in 2nd array
def countElements(arr1, arr2, m, n):
# sort the 2nd array
arr2.sort()

# for each element in first array
for i in range(m):
    # last index of largest element
    # smaller than or equal to x
    index = bin_search(arr2, n, arr1[i])
    # required count for the element arr1[i]
    print(index + 1)

# driver program to test above function
arr1 = [1, 2, 3, 4, 7, 9]
arr2 = [0, 1, 2, 1, 1, 4]
m = len(arr1)
n = len(arr2)
countElements(arr1, arr2, m, n)

#This code is contributed by Aditi Sharma
```

C#

```
 // C# implementation of For each element
// in 1st array count elements less than
// or equal to it in 2nd array
using System;

public class GFG
{

    // method returns the index of
    // largest element smaller than
    // equal to 'x' in 'arr'. For
    // duplicates it returns the last
    // index of occurrence of required
    // element. If no such element
    // exits then it returns -1.
```

```
static int binary_search(int []arr,
                 int l, int h, int x)
{
    while (l <= h)
    {
        int mid = (l+h) / 2;

        // if 'x' is greater than or
        // equal to arr[mid], then
        // search in arr[mid+1...h]
        if (arr[mid] <= x)
            l = mid + 1;

        // else search in
        // arr[l...mid-1]
        else
            h = mid - 1;
    }

    // required index
    return h;
}

// method to count for each element
// in 1st array, elements less than
// or equal to it in 2nd array
static void countEleLessThanOrEqual(
             int []arr1, int []arr2,
                       int m, int n)
{

    // sort the 2nd array
    Array.Sort(arr2);

    // for each element of 1st array
    for (int i=0; i<m; i++)
    {
        // last index of largest
        // element smaller than or
        // equal to x
        int index = binary_search(
               arr2, 0, n-1, arr1[i]);

        // required count for the
        // element arr1[i]
        Console.Write((index+1) + " ");
    }
}
```

```
    // Driver method
    public static void Main()
    {
        int []arr1 = {1, 2, 3, 4, 7, 9};
        int []arr2= {0, 1, 2, 1, 1, 4};

        countEleLessThanOrEqual(arr1,
           arr2, arr1.Length, arr2.Length);
    }
}

// This code is contributed by Sam007.
```

PHP

```
 <?php
// php implementation of For each element
// in 1st array count elements less than
// or equal to it in 2nd array

// function returns the index of largest
// element smaller than equal to 'x' in
// 'arr'. For duplicates it returns the
// last index of occurrence of required
// element. If no such element exits then
// it returns -1.
function binary_search($arr, $l, $h, $x)
{
    while ($l <= $h)
    {
        $mid = (floor($l+$h) / 2);

        // if 'x' is greater than or
        // equal to arr[mid], then
        // search in arr[mid+1...h]
        if ($arr[$mid] <= $x)
            $l = $mid + 1;

        // else search in arr[l...mid-1]
        else
            $h = $mid - 1;
    }

    // required index
    return $h;
}
```

```
// function to count for each element
// in 1st array, elements less than or
// equal to it in 2nd array
function countEleLessThanOrEqual($arr1,
                          $arr2, $m,$n)
{
    // sort the 2nd array
    sort($arr2); sort($arr2, $n);

    // for each element of 1st array
    for ($i = 0; $i < $m; $i++)
    {
        // last index of largest element
        // smaller than or equal to x
        $index = binary_search($arr2, 0,
                        $n-1, $arr1[$i]);

        // required count for the
        // element arr1[i]
        echo ($index+1) , " ";
    }
}

// Driver program to test above
$arr1 = array(1, 2, 3, 4, 7, 9);
$arr2 = array(0, 1, 2, 1, 1, 4);
$m = sizeof($arr1) / sizeof($arr1[0]);
$n = sizeof($arr2) / sizeof($arr2[0]);
countEleLessThanOrEqual($arr1, $arr2, $m, $n);

//This code is contributed by nitin mittal.
?>
```

Output:

```
4 5 5 6 6 6
```

Improved By : Sam007, nitin mittal

Source

https://www.geeksforgeeks.org/element-1st-array-count-elements-less-equal-2nd-array/

Chapter 117

Front and Back Search in unsorted array

Front and Back Search in unsorted array - GeeksforGeeks

Given an unsorted array of integers and an element x, find if x is present in array using Front and Back search.

Examples :

```
Input : arr[] = {10, 20, 80, 30, 60, 50,
                 110, 100, 130, 170}
          x = 110;
Output : Yes

Input : arr[] = {10, 20, 80, 30, 60, 50,
                 110, 100, 130, 170}
           x = 175;
Output : No
```

A simple solution is to performlinear search. The linear search algorithm always results in worst case complexity of O(n) when element to be searched is last element in the array. Front and Back search algorithm for finding element with value x works the following way:

1. Initialize indexes *front* and *back* pointing to first and last element respectively of the array.
2. If *front* is greater than *rear*, return false.
3. Check the element x at front and rear index.
4. If element x is found return true.
5. Else increment *front* and decrement *rear* and go to step 2.

Key Points:

- The worst case complexity is O(n/2) (equivalent to O(n)) when element is in the middle or not present in the array.
- The best case complexity is O(1) when element is first or last element in the array.

C++

```
 // CPP program to implement front and back
// search
#include<iostream>
using namespace std;

bool search(int arr[], int n, int x)
{
    // Start searching from both ends
    int front = 0, back = n - 1;

    // Keep searching while two indexes
    // do not cross.
    while (front <= back)
    {
        if (arr[front] == x || arr[back] == x)
          return true;
        front++;
        back--;
    }
    return false;
}

int main()
{
   int arr[] = {10, 20, 80, 30, 60, 50,
                     110, 100, 130, 170};
   int x = 130;
   int n = sizeof(arr)/sizeof(arr[0]);
   if (search(arr, n, x))
      cout << "Yes";
   else
      cout << "No";
   return 0;
}
```

Java

```
 // Java program to implement front and back
// search
class GFG {
```

```
    static boolean search(int arr[], int n, int x)
    {

        // Start searching from both ends
        int front = 0, back = n - 1;

        // Keep searching while two indexes
        // do not cross.
        while (front <= back)
        {
            if (arr[front] == x || arr[back] == x)
                return true;
            front++;
            back--;
        }

        return false;
    }

    // driver code
    public static void main (String[] args)
    {
        int arr[] = {10, 20, 80, 30, 60, 50,
                        110, 100, 130, 170};
        int x = 130;
        int n = arr.length;

        if (search(arr, n, x))
            System.out.print("Yes");
        else
            System.out.print("No");
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python program to implement
# front and back search

def search(arr, n, x):

    # Start searching from both ends
    front = 0; back = n - 1

    # Keep searching while two
    # indexes do not cross.
```

```
    while (front <= back):

        if (arr[front] == x or arr[back] == x):
            return True
        front += 1
        back -= 1

    return False

# Driver code
arr = [10, 20, 80, 30, 60,
       50, 110, 100, 130, 170]
x = 130
n = len(arr)

if (search(arr, n, x)):
    print("Yes")
else:
    print("No")

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to implement front and back
// search
using System;

public class GFG {

    static bool search(int []arr, int n, int x)
    {

        // Start searching from both ends
        int front = 0, back = n - 1;

        // Keep searching while two indexes
        // do not cross.
        while (front <= back)
        {
            if (arr[front] == x || arr[back] == x)
                return true;
            front++;
            back--;
        }

        return false;
    }
```

```
    static public void Main ()
    {
        int []arr = {10, 20, 80, 30, 60, 50,
                    110, 100, 130, 170};
        int x = 130;
        int n = arr.Length;

        if (search(arr, n, x))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to implement front and back
// search

function search($arr, $n, $x)
{

    // Start searching from both ends
    $front = 0;
    $back = $n - 1;

    // Keep searching while two
    // indexes do not cross.
    while ($front <= $back)
    {
        if ($arr[$front] == $x ||
            $arr[$back] == $x)
        return true;
        $front++;
        $back--;
    }
    return false;
}

    // Driver Code
    $arr = array(10, 20, 80, 30, 60, 50,
                    110, 100, 130, 170);
    $x = 130;
    $n = sizeof($arr);
```

```
    if (search($arr, $n, $x))
        echo "Yes";
    else
        echo "No";

// This code is contributed by aj_36
?>
```

Output:

```
Yes
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/front-and-back-search-in-unsorted-array/

Chapter 118

Given a sorted and rotated array, find if there is a pair with a given sum

Given a sorted and rotated array, find if there is a pair with a given sum - GeeksforGeeks

Given an array that is sorted and then rotated around an unknown point. Find if array has a pair with given sum 'x'. It may be assumed that all elements in array are distinct.

Examples :

```
Input: arr[] = {11, 15, 6, 8, 9, 10}, x = 16
Output: true
There is a pair (6, 10) with sum 16

Input: arr[] = {11, 15, 26, 38, 9, 10}, x = 35
Output: true
There is a pair (26, 9) with sum 35

Input: arr[] = {11, 15, 26, 38, 9, 10}, x = 45
Output: false
There is no pair with sum 45.
```

We have discussed a O(n) solution for a sorted array (See steps 2, 3 and 4 of Method 1). We can extend this solution for rotated array as well. The idea is to first find the maximum element in array which is the pivot point also and the element just before maximum is the minimum element. Once we have indexes maximum and minimum elements, we use similar meet in middle algorithm (as discussed here in method 1) to find if there is a pair. The only thing new here is indexes are incremented and decremented in rotational manner using modular arithmetic.

Following is the implementation of above idea.

C++

```
 // C++ program to find a pair with a given sum in a sorted and
// rotated array
#include<iostream>
using namespace std;

// This function returns true if arr[0..n-1] has a pair
// with sum equals to x.
bool pairInSortedRotated(int arr[], int n, int x)
{
    // Find the pivot element
    int i;
    for (i=0; i<n-1; i++)
        if (arr[i] > arr[i+1])
            break;
    int l = (i+1)%n;  // l is now index of minimum element
    int r = i;        // r is now index of maximum element

    // Keep moving either l or r till they meet
    while (l != r)
    {
         // If we find a pair with sum x, we return true
         if (arr[l] + arr[r] == x)
              return true;

         // If current pair sum is less, move to the higher sum
         if (arr[l] + arr[r] < x)
              l = (l + 1)%n;
         else  // Move to the lower sum side
              r = (n + r - 1)%n;
    }
    return false;
}

/* Driver program to test above function */
int main()
{
    int arr[] = {11, 15, 6, 8, 9, 10};
    int sum = 16;
    int n = sizeof(arr)/sizeof(arr[0]);

    if (pairInSortedRotated(arr, n, sum))
        cout << "Array has two elements with sum 16";
    else
        cout << "Array doesn't have two elements with sum 16 ";
```

```
    return 0;
}
```

Java

```
 // Java program to find a pair with a given
// sum in a sorted and rotated array
class PairInSortedRotated
{
    // This function returns true if arr[0..n-1]
    // has a pair with sum equals to x.
    static boolean pairInSortedRotated(int arr[],
                                    int n, int x)
    {
        // Find the pivot element
        int i;
        for (i = 0; i < n - 1; i++)
            if (arr[i] > arr[i+1])
                break;

        int l = (i + 1) % n; // l is now index of
                             // minimum element

        int r = i;       // r is now index of maximum
                         //element

        // Keep moving either l or r till they meet
        while (l != r)
        {
             // If we find a pair with sum x, we
             // return true
             if (arr[l] + arr[r] == x)
                  return true;

             // If current pair sum is less, move
             // to the higher sum
             if (arr[l] + arr[r] < x)
                  l = (l + 1) % n;

             else  // Move to the lower sum side
                  r = (n + r - 1) % n;
        }
        return false;
    }

    /* Driver program to test above function */
    public static void main (String[] args)
```

```
    {
        int arr[] = {11, 15, 6, 8, 9, 10};
        int sum = 16;
        int n = arr.length;

        if (pairInSortedRotated(arr, n, sum))
            System.out.print("Array has two elements" +
                             " with sum 16");
        else
            System.out.print("Array doesn't have two" +
                             " elements with sum 16 ");
    }
}
/*This code is contributed by Prakriti Gupta*/
```

Python3

```
 # Python3 program to find a
# pair with a given sum in
# a sorted and rotated array


# This function returns True
# if arr[0..n-1] has a pair
# with sum equals to x.
def pairInSortedRotated( arr, n, x ):

    # Find the pivot element
    for i in range(0, n - 1):
        if (arr[i] > arr[i + 1]):
            break;

    # l is now index of minimum element
    l = (i + 1) % n
    # r is now index of maximum element
    r = i

    # Keep moving either l
    # or r till they meet
    while (l != r):

        # If we find a pair with
        # sum x, we return True
        if (arr[l] + arr[r] == x):
            return True;

        # If current pair sum is less,
        # move to the higher sum
```

```
        if (arr[l] + arr[r] < x):
            l = (l + 1) % n;
        else:

            # Move to the lower sum side
            r = (n + r - 1) % n;

    return False;


# Driver program to test above function
arr = [11, 15, 26, 38, 9, 10]
sum = 16
n = len(arr)

if (pairInSortedRotated(arr, n, sum)):
    print ("Array has two elements with sum 16")
else:
    print ("Array doesn't have two elements with sum 16 ")


# This article contributed by saloni1297
```

C#

```
 // C# program to find a pair with a given
// sum in a sorted and rotated array
using System;

class PairInSortedRotated
{
    // This function returns true if arr[0..n-1]
    // has a pair with sum equals to x.
    static bool pairInSortedRotated(int []arr,
                                    int n, int x)
    {
        // Find the pivot element
        int i;
        for (i = 0; i < n - 1; i++)
            if (arr[i] > arr[i + 1])
                break;

        // l is now index of minimum element
        int l = (i + 1) % n;

        // r is now index of maximum element
        int r = i;
```

```
        // Keep moving either l or r till they meet
        while (l != r)
        {
            // If we find a pair with sum x, we
            // return true
            if (arr[l] + arr[r] == x)
                return true;

            // If current pair sum is less,
            // move to the higher sum
            if (arr[l] + arr[r] < x)
                l = (l + 1) % n;

            // Move to the lower sum side
            else
                r = (n + r - 1) % n;
        }
        return false;
    }

    // Driver Code
    public static void Main ()
    {
        int []arr = {11, 15, 6, 8, 9, 10};
        int sum = 16;
        int n = arr.Length;

        if (pairInSortedRotated(arr, n, sum))
            Console.WriteLine("Array has two elements" +
                                    " with sum 16");
        else
        Console.WriteLine("Array doesn't have two" +
                          " elements with sum 16 ");
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find a pair
// with a given sum in a
// sorted and rotated array

// This function returns true
// if arr[0..n-1] has a pair
// with sum equals to x.
```

```
function pairInSortedRotated($arr, $n, $x)
{
    // Find the pivot element
    $i;
    for ($i = 0; $i < $n - 1; $i++)
        if ($arr[$i] > $arr[$i + 1])
            break;

    // l is now index of
    // minimum element
    $l = ($i + 1) % $n;

    // r is now index of
    // maximum element
    $r = $i;

    // Keep moving either l
    // or r till they meet
    while ($l != $r)
    {
        // If we find a pair with
        // sum x, we return true
        if ($arr[$l] + $arr[$r] == $x)
            return true;

        // If current pair sum is
        // less, move to the higher sum
        if ($arr[$l] + $arr[$r] < $x)
            $l = ($l + 1) % $n;

        // Move to the lower sum side
        else
            $r = ($n + $r - 1) % $n;
    }
    return false;
}

// Driver Code
$arr = array(11, 15, 6, 8, 9, 10);
$sum = 16;
$n = sizeof($arr);

if (pairInSortedRotated($arr, $n, $sum))
    echo "Array has two elements ".
                    "with sum 16";
else
    echo "Array doesn't have two ".
```

```
          "elements with sum 16 ";

// This code is contributed by aj_36
?>
```

Output :

```
Array has two elements with sum 16
```

Time complexity of the above solution is O(n). The step to find the pivot can be optimized to O(Logn) using the Binary Search approach discussed here.

How to count all pairs having sum x?
The stepwise algo is:

1. Find the pivot element of sorted and rotated array. The pivot element is the maximum element in the array. The minimum element will be adjacent to it.
2. Use two pointers with left pointer pointing to minimum element and right pointer pointing to maximum element.
3. Find sum of the elements pointed by both the pointers.
4. If sum is equal to x, then increment count. If sum is less than x, then to increase sum move left pointer to next position by incrementing it in rotational manner. If sum is greater than x, then to decrease sum move right pointer to next position by decrementing it in rotational manner.
5. Repeat step 3 and 4 until left pointer is not equal to right pointer or until left pointer is not equal to right pointer – 1.
6. Print final count.

Below is implementation of above algorithm:

C++

```
 // C++ program to find number of pairs with
// a given sum in a sorted and rotated array.
#include<bits/stdc++.h>
using namespace std;

// This function returns count of number of pairs
// with sum equals to x.
int pairsInSortedRotated(int arr[], int n, int x)
{
    // Find the pivot element. Pivot element
    // is maximum element of array.
    int i;
    for (i = 0; i < n-1; i++)
        if (arr[i] > arr[i+1])
            break;
```

```
    // l is index of minimum element.
    int l = (i + 1) % n;

    // r is index of maximum element.
    int r = i;

    // Variable to store count of number
    // of pairs.
    int cnt = 0;

    // Find sum of pair formed by arr[l] and
    // and arr[r] and update l, r and cnt
    // accordingly.
    while (l != r)
    {
        // If we find a pair with sum x, then
        // increment cnt, move l and r to
        // next element.
        if (arr[l] + arr[r] == x){
            cnt++;

            // This condition is required to
            // be checked, otherwise l and r
            // will cross each other and loop
            // will never terminate.
            if(l == (r - 1 + n) % n){
                return cnt;
            }

            l = (l + 1) % n;
            r = (r - 1 + n) % n;
        }

        // If current pair sum is less, move to
        // the higher sum side.
        else if (arr[l] + arr[r] < x)
            l = (l + 1) % n;

        // If current pair sum is greater, move
        // to the lower sum side.
        else
            r = (n + r - 1)%n;
    }

    return cnt;
}
```

```
/* Driver program to test above function */
int main()
{
    int arr[] = {11, 15, 6, 7, 9, 10};
    int sum = 16;
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << pairsInSortedRotated(arr, n, sum);

    return 0;
}
```

Java

```
 // Java program to find
// number of pairs with
// a given sum in a sorted
// and rotated array.
import java.io.*;

class GFG
{

// This function returns
// count of number of pairs
// with sum equals to x.
static int pairsInSortedRotated(int arr[],
                                int n, int x)
{
    // Find the pivot element.
    // Pivot element is maximum
    // element of array.
    int i;
    for (i = 0; i < n - 1; i++)
        if (arr[i] > arr[i + 1])
            break;

    // l is index of
    // minimum element.
    int l = (i + 1) % n;

    // r is index of
    // maximum element.
    int r = i;

    // Variable to store
    // count of number
    // of pairs.
```

```
    int cnt = 0;

    // Find sum of pair
    // formed by arr[l]
    // and arr[r] and
    // update l, r and
    // cnt accordingly.
    while (l != r)
    {
        // If we find a pair with
        // sum x, then increment
        // cnt, move l and r to
        // next element.
        if (arr[l] + arr[r] == x)
        {
            cnt++;

            // This condition is required
            // to be checked, otherwise
            // l and r will cross each
            // other and loop will never
            // terminate.
            if(l == (r - 1 + n) % n)
            {
                return cnt;
            }

            l = (l + 1) % n;
            r = (r - 1 + n) % n;
        }

        // If current pair sum
        // is less, move to
        // the higher sum side.
        else if (arr[l] + arr[r] < x)
            l = (l + 1) % n;

        // If current pair sum
        // is greater, move
        // to the lower sum side.
        else
            r = (n + r - 1) % n;
    }

    return cnt;
}

// Driver Code
```

```
public static void main (String[] args)
{
    int arr[] = {11, 15, 6, 7, 9, 10};
    int sum = 16;
    int n = arr.length;

    System.out.println(
            pairsInSortedRotated(arr, n, sum));
}
}

// This code is contributed by ajit
```

Python 3

```
 # Python program to find
# number of pairs with
# a given sum in a sorted
# and rotated array.

# This function returns
# count of number of pairs
# with sum equals to x.
def pairsInSortedRotated(arr, n, x):

    # Find the pivot element.
    # Pivot element is maximum
    # element of array.
    for i in range(n):
        if arr[i] > arr[i + 1]:
            break

    # l is index of
    # minimum element.
    l = (i + 1) % n

    # r is index of
    # maximum element.
    r = i

    # Variable to store
    # count of number
    # of pairs.
    cnt = 0

    # Find sum of pair
    # formed by arr[l]
    # and arr[r] and
```

```
    # update l, r and
    # cnt accordingly.
    while (l != r):

        # If we find a pair
        # with sum x, then
        # increment cnt, move
        # l and r to next element.
        if arr[l] + arr[r] == x:
            cnt += 1

            # This condition is
            # required to be checked,
            # otherwise l and r will
            # cross each other and
            # loop will never terminate.
            if l == (r - 1 + n) % n:
                return cnt

            l = (l + 1) % n
            r = (r - 1 + n) % n

        # If current pair sum
        # is less, move to
        # the higher sum side.
        elif arr[l] + arr[r] < x:
            l = (l + 1) % n

        # If current pair sum
        # is greater, move to
        # the lower sum side.
        else:
            r = (n + r - 1) % n

    return cnt

# Driver Code
arr = [11, 15, 6, 7, 9, 10]
s = 16

print(pairsInSortedRotated(arr, 6, s))

# This code is contributed
# by ChitraNayal
```

C#

```
 // C# program to find
```

```
// number of pairs with
// a given sum in a sorted
// and rotated array.
using System;

class GFG
{

// This function returns
// count of number of pairs
// with sum equals to x.
static int pairsInSortedRotated(int []arr,
                                int n, int x)
{
    // Find the pivot element.
    // Pivot element is maximum
    // element of array.
    int i;
    for (i = 0; i < n - 1; i++)
        if (arr[i] > arr[i + 1])
            break;

    // l is index of
    // minimum element.
    int l = (i + 1) % n;

    // r is index of
    // maximum element.
    int r = i;

    // Variable to store
    // count of number
    // of pairs.
    int cnt = 0;

    // Find sum of pair
    // formed by arr[l]
    // and arr[r] and
    // update l, r and
    // cnt accordingly.
    while (l != r)
    {
        // If we find a pair with
        // sum x, then increment
        // cnt, move l and r to
        // next element.
        if (arr[l] + arr[r] == x)
        {
```

```
            cnt++;

            // This condition is required
            // to be checked, otherwise
            // l and r will cross each
            // other and loop will never
            // terminate.
            if(l == (r - 1 + n) % n)
            {
                return cnt;
            }

            l = (l + 1) % n;
            r = (r - 1 + n) % n;
        }

        // If current pair sum
        // is less, move to
        // the higher sum side.
        else if (arr[l] + arr[r] < x)
            l = (l + 1) % n;

        // If current pair sum
        // is greater, move
        // to the lower sum side.
        else
            r = (n + r - 1) % n;
    }

    return cnt;
}

// Driver Code
static public void Main ()
{
    int []arr = {11, 15, 6, 7, 9, 10};
    int sum = 16;
    int n = arr.Length;

    Console.WriteLine(
            pairsInSortedRotated(arr, n, sum));
}
}

// This code is contributed by akt_mit
```

PHP

```
 <?php
// PHP program to find number
// of pairs with a given sum
// in a sorted and rotated array.

// This function returns count
// of number of pairs with sum
// equals to x.
function pairsInSortedRotated($arr,
                              $n, $x)
{
    // Find the pivot element.
    // Pivot element is maximum
    // element of array.
    $i;
    for ($i = 0; $i < $n - 1; $i++)
        if ($arr[$i] > $arr[$i + 1])
            break;

    // l is index of
    // minimum element.
    $l = ($i + 1) % $n;

    // r is index of
    // maximum element.
    $r = $i;

    // Variable to store
    // count of number
    // of pairs.
    $cnt = 0;

    // Find sum of pair formed
    // by arr[l] and arr[r] and
    // update l, r and cnt
    // accordingly.
    while ($l != $r)
    {
        // If we find a pair with
        // sum x, then increment
        // cnt, move l and r to
        // next element.
        if ($arr[$l] + $arr[$r] == $x)
        {
            $cnt++;

            // This condition is required
            // to be checked, otherwise l
```

```
            // and r will cross each other
            // and loop will never terminate.
            if($l == ($r - 1 + $n) % $n)
            {
                return $cnt;
            }

            $l = ($l + 1) % $n;
            $r = ($r - 1 + $n) % $n;
        }

        // If current pair sum
        // is less, move to
        // the higher sum side.
        else if ($arr[$l] + $arr[$r] < $x)
            $l = ($l + 1) % $n;

        // If current pair sum
        // is greater, move to
        // the lower sum side.
        else
            $r = ($n + $r - 1) % $n;
    }

    return $cnt;
}

// Driver Code
$arr = array(11, 15, 6,
             7, 9, 10);
$sum = 16;
$n = sizeof($arr) / sizeof($arr[0]);

echo pairsInSortedRotated($arr,
                          $n, $sum);

// This code is contributed by ajit
?>
```

Output:

```
2
```

Time Complexity: O(n)
Auxiliary Space: O(1)
This method is suggested by Nikhil Jindal.

Exercise:
1) Extend the above solution to work for arrays with duplicates allowed.

This article is contributed by **Himanshu Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : jit_t, nik1996, ChitraNayal

Source

https://www.geeksforgeeks.org/given-a-sorted-and-rotated-array-find-if-there-is-a-pair-with-a-given-sum/

Chapter 119

Given a sorted array and a number x, find the pair in array whose sum is closest to x

Given a sorted array and a number x, find the pair in array whose sum is closest to x - GeeksforGeeks

Given a sorted array and a number x, find a pair in array whose sum is closest to x.

Examples:

```
Input: arr[] = {10, 22, 28, 29, 30, 40}, x = 54
Output: 22 and 30

Input: arr[] = {1, 3, 4, 7, 10}, x = 15
Output: 4 and 10
```

A simple solution is to consider every pair and keep track of closest pair (absolute difference between pair sum and x is minimum). Finally print the closest pair. Time complexity of this solution is $O(n^2)$

An efficient solution can find the pair in O(n) time. The idea is similar to method 2 of thispost. Following is detailed algorithm.

```
1) Initialize a variable diff as infinite (Diff is used to store the
   difference between pair and x).  We need to find the minimum diff.
2) Initialize two index variables l and r in the given sorted array.
       (a) Initialize first to the leftmost index:  l = 0
       (b) Initialize second  the rightmost index:  r = n-1
3) Loop while l < r.
```

```
(a) If  abs(arr[l] + arr[r] - sum) < diff  then
    update diff and result
(b) Else if(arr[l] + arr[r] <  sum )  then l++
(c) Else r--
```

Following is the implementation of above algorithm.

C++

```
 // Simple C++ program to find the pair with sum closest to a given no.
#include <iostream>
#include <climits>
#include <cstdlib>
using namespace std;

// Prints the pair with sum closest to x
void printClosest(int arr[], int n, int x)
{
    int res_l, res_r;  // To store indexes of result pair

    // Initialize left and right indexes and difference between
    // pair sum and x
    int l = 0, r = n-1, diff = INT_MAX;

    // While there are elements between l and r
    while (r > l)
    {
       // Check if this pair is closer than the closest pair so far
       if (abs(arr[l] + arr[r] - x) < diff)
       {
           res_l = l;
           res_r = r;
           diff = abs(arr[l] + arr[r] - x);
       }

       // If this pair has more sum, move to smaller values.
       if (arr[l] + arr[r] > x)
           r--;
       else // Move to larger values
           l++;
    }

    cout <<" The closest pair is " << arr[res_l] << " and " << arr[res_r];
}

// Driver program to test above functions
int main()
{
```

```
    int arr[] =  {10, 22, 28, 29, 30, 40}, x = 54;
    int n = sizeof(arr)/sizeof(arr[0]);
    printClosest(arr, n, x);
    return 0;
}
```

Java

```
 // Java program to find pair with sum closest to x
import java.io.*;
import java.util.*;
import java.lang.Math;

class CloseSum {

    // Prints the pair with sum cloest to x
    static void printClosest(int arr[], int n, int x)
    {
        int res_l=0, res_r=0;  // To store indexes of result pair

        // Initialize left and right indexes and difference between
        // pair sum and x
        int l = 0, r = n-1, diff = Integer.MAX_VALUE;

        // While there are elements between l and r
        while (r > l)
        {
            // Check if this pair is closer than the closest pair so far
            if (Math.abs(arr[l] + arr[r] - x) < diff)
            {
               res_l = l;
               res_r = r;
               diff = Math.abs(arr[l] + arr[r] - x);
            }

            // If this pair has more sum, move to smaller values.
            if (arr[l] + arr[r] > x)
               r--;
            else // Move to larger values
               l++;
        }

    System.out.println(" The closest pair is "+arr[res_l]+" and "+ arr[res_r]);
}


    // Driver program to test above function
    public static void main(String[] args)
```

```
    {
        int arr[] =  {10, 22, 28, 29, 30, 40}, x = 54;
        int n = arr.length;
        printClosest(arr, n, x);
    }
}
/*This code is contributed by Devesh Agrawal*/
```

Python3

```
 # Python3 program to find the pair
# with sum
# closest to a given no.

# A sufficiently large value greater
# than any
# element in the input array
MAX_VAL = 1000000000


#Prints the pair with sum closest to x

def printClosest(arr, n, x):

    # To store indexes of result pair
    res_l, res_r = 0, 0

    #Initialize left and right indexes
    # and difference between
    # pair sum and x
    l, r, diff = 0, n-1, MAX_VAL

    # While there are elements between l and r
    while r > l:
        # Check if this pair is closer than the
        # closest pair so far
        if abs(arr[l] + arr[r] - x) < diff:
            res_l = l
            res_r = r
            diff = abs(arr[l] + arr[r] - x)

        if arr[l] + arr[r] > x:
        # If this pair has more sum, move to
        # smaller values.
            r -= 1
        else:
        # Move to larger values
            l += 1
```

```
    print('The closest pair is {} and {}'
          .format(arr[res_l], arr[res_r]))


# Driver code to test above
if __name__ == "__main__":
    arr = [10, 22, 28, 29, 30, 40]
    n = len(arr)
    x=54
    printClosest(arr, n, x)

# This code is contributed by Tuhin Patra
```

C#

```
 // C# program to find pair with sum closest to x
using System;

class GFG {

    // Prints the pair with sum cloest to x
    static void printClosest(int []arr, int n, int x)
    {

        // To store indexes of result pair
        int res_l = 0, res_r = 0;

        // Initialize left and right indexes and
        // difference between pair sum and x
        int l = 0, r = n-1, diff = int.MaxValue;

        // While there are elements between l and r
        while (r > l)
        {

            // Check if this pair is closer than the
            // closest pair so far
            if (Math.Abs(arr[l] + arr[r] - x) < diff)
            {
                res_l = l;
                res_r = r;
                diff = Math.Abs(arr[l] + arr[r] - x);
            }

            // If this pair has more sum, move to
            // smaller values.
            if (arr[l] + arr[r] > x)
```

```
            r--;
            else // Move to larger values
            l++;
        }

        Console.Write(" The closest pair is " +
                  arr[res_l] + " and " + arr[res_r]);
    }

    // Driver program to test above function
    public static void Main()
    {
        int []arr = {10, 22, 28, 29, 30, 40};
        int x = 54;
        int n = arr.Length;

        printClosest(arr, n, x);
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// Simple PHP program to find the
// pair with sum closest to a
// given no.

// Prints the pair with
// sum closest to x
function printClosest($arr, $n, $x)
{

    // To store indexes
    // of result pair
    $res_l;
    $res_r;

    // Initialize left and right
    // indexes and difference between
    // pair sum and x
    $l = 0;
    $r = $n - 1;
    $diff = PHP_INT_MAX;

    // While there are elements
    // between l and r
```

```
    while ($r > $l)
    {

        // Check if this pair is closer
        // than the closest pair so far
        if (abs($arr[$l] + $arr[$r] - $x) <
                                    $diff)
        {
            $res_l = $l;
            $res_r = $r;
            $diff = abs($arr[$l] + $arr[$r] - $x);
        }

        // If this pair has more sum,
        // move to smaller values.
        if ($arr[$l] + $arr[$r] > $x)
            $r--;

        // Move to larger values
        else
            $l++;
    }

    echo " The closest pair is "
         , $arr[$res_l] ," and "
         , $arr[$res_r];
}

    // Driver Code
    $arr = array(10, 22, 28, 29, 30, 40);
    $x = 54;
    $n = count($arr);
    printClosest($arr, $n, $x);

// This code is contributed by anuj_67.
?>
```

Output:

```
 The closest pair is 22 and 30
```

This article is contributed by **Harsh**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal, vt_m, TuhinPatra

Source

https://www.geeksforgeeks.org/given-sorted-array-number-x-find-pair-array-whose-sum-closest-x/

Chapter 120

Given an array of of size n and a number k, find all elements that appear more than n/k times

Given an array of of size n and a number k, find all elements that appear more than n/k times - GeeksforGeeks

Given an array of size n, find all elements in array that appear more than n/k times. For example, if the input arrays is {3, 1, 2, 2, 1, 2, 3, 3} and k is 4, then the output should be [2, 3]. Note that size of array is 8 (or n = 8), so we need to find all elements that appear more than 2 (or 8/4) times. There are two elements that appear more than two times, 2 and 3.

A **simple method** is to pick all elements one by one. For every picked element, count its occurrences by traversing the array, if count becomes more than n/k, then print the element. Time Complexity of this method would be $O(n^2)$.

A better solution is to **use sorting**. First, sort all elements using a O(nLogn) algorithm. Once the array is sorted, we can find all required elements in a linear scan of array. So overall time complexity of this method is O(nLogn) + O(n) which is O(nLogn).

Following is an interesting **O(nk) solution:**
We can solve the above problem in O(nk) time using O(k-1) extra space. Note that there can never be more than k-1 elements in output (Why?). There are mainly three steps in this algorithm.

1) Create a temporary array of size (k-1) to store elements and their counts (The output elements are going to be among these k-1 elements). Following is structure of temporary array elements.

```
struct eleCount {
    int element;
```

```
    int count;
};
struct eleCount temp[];
```

This step takes O(k) time.

2) Traverse through the input array and update temp[] (add/remove an element or increase/decrease count) for every traversed element. The array temp[] stores potential (k-1) candidates at every step. This step takes O(nk) time.

3) Iterate through final (k-1) potential candidates (stored in temp[]). or every element, check if it actually has count more than n/k. This step takes O(nk) time.

The main step is step 2, how to maintain (k-1) potential candidates at every point? The steps used in step 2 are like famous game: Tetris. We treat each number as a piece in Tetris, which falls down in our temporary array temp[]. Our task is to try to keep the same number stacked on the same column (count in temporary array is incremented).

```
Consider k = 4, n = 9
Given array: 3 1 2 2 2 1 4 3 3

i = 0
         3 _ _
temp[] has one element, 3 with count 1

i = 1
         3 1 _
temp[] has two elements, 3 and 1 with
counts 1 and 1 respectively

i = 2
         3 1 2
temp[] has three elements, 3, 1 and 2 with
counts as 1, 1 and 1 respectively.

i = 3
         - - 2
         3 1 2
temp[] has three elements, 3, 1 and 2 with
counts as 1, 1 and 2 respectively.

i = 4
         - - 2
         - - 2
         3 1 2
temp[] has three elements, 3, 1 and 2 with
counts as 1, 1 and 3 respectively.
```

```
i = 5
         - - 2
         - 1 2
         3 1 2
temp[] has three elements, 3, 1 and 2 with
counts as 1, 2 and 3 respectively.
```

Now the question arises, what to do when temp[] is full and we see a new element – we remove the bottom row from stacks of elements, i.e., we decrease count of every element by 1 in temp[]. We ignore the current element.

```
i = 6
         - - 2
         - 1 2
temp[] has two elements, 1 and 2 with
counts as 1 and 2 respectively.

i = 7
           - 2
         3 1 2
temp[] has three elements, 3, 1 and 2 with
counts as 1, 1 and 2 respectively.

i = 8
         3 - 2
         3 1 2
temp[] has three elements, 3, 1 and 2 with
counts as 2, 1 and 2 respectively.
```

Finally, we have at most k-1 numbers in temp[]. The elements in temp are {3, 1, 2}. Note that the counts in temp[] are useless now, the counts were needed only in step 2. Now we need to check whether the actual counts of elements in temp[] are more than n/k (9/4) or not. The elements 3 and 2 have counts more than 9/4. So we print 3 and 2.

Note that the algorithm doesn't miss any output element. There can be two possibilities, many occurrences are together or spread across the array. If occurrences are together, then count will be high and won't become 0. If occurrences are spread, then the element would come again in temp[]. Following is C++ implementation of above algorithm.

```
 // A C++ program to print elements with count more than n/k
#include<iostream>
using namespace std;

// A structure to store an element and its current count
struct eleCount
{
    int e;  // Element
    int c;  // Count
```

```
};

// Prints elements with more than n/k occurrences in arr[] of
// size n. If there are no such elements, then it prints nothing.
void moreThanNdK(int arr[], int n, int k)
{
    // k must be greater than 1 to get some output
    if (k < 2)
       return;

    /* Step 1: Create a temporary array (contains element
       and count) of size k-1. Initialize count of all
       elements as 0 */
    struct eleCount temp[k-1];
    for (int i=0; i<k-1; i++)
        temp[i].c = 0;

    /* Step 2: Process all elements of input array */
    for (int i = 0; i < n; i++)
    {
        int j;

        /* If arr[i] is already present in
           the element count array, then increment its count */
        for (j=0; j<k-1; j++)
        {
            if (temp[j].e == arr[i])
            {
                 temp[j].c += 1;
                 break;
            }
        }

        /* If arr[i] is not present in temp[] */
        if (j == k-1)
        {
            int l;

            /* If there is position available in temp[], then place
              arr[i] in the first available position and set count as 1*/
            for (l=0; l<k-1; l++)
            {
                if (temp[l].c == 0)
                {
                    temp[l].e = arr[i];
                    temp[l].c = 1;
                    break;
                }
```

```
            }

            /* If all the position in the temp[] are filled, then
               decrease count of every element by 1 */
            if (l == k-1)
                for (l=0; l<k; l++)
                    temp[l].c -= 1;
        }
    }

    /*Step 3: Check actual counts of potential candidates in temp[]*/
    for (int i=0; i<k-1; i++)
    {
        // Calculate actual count of elements
        int ac = 0;  // actual count
        for (int j=0; j<n; j++)
            if (arr[j] == temp[i].e)
                ac++;

        // If actual count is more than n/k, then print it
        if (ac > n/k)
           cout << "Number:" << temp[i].e
                << " Count:" << ac << endl;
    }
}

/* Driver program to test above function */
int main()
{
    cout << "First Test\n";
    int arr1[] = {4, 5, 6, 7, 8, 4, 4};
    int size = sizeof(arr1)/sizeof(arr1[0]);
    int k = 3;
    moreThanNdK(arr1, size, k);

    cout << "\nSecond Test\n";
    int arr2[] = {4, 2, 2, 7};
    size = sizeof(arr2)/sizeof(arr2[0]);
    k = 3;
    moreThanNdK(arr2, size, k);

    cout << "\nThird Test\n";
    int arr3[] = {2, 7, 2};
    size = sizeof(arr3)/sizeof(arr3[0]);
    k = 2;
    moreThanNdK(arr3, size, k);

    cout << "\nFourth Test\n";
```

```
    int arr4[] = {2, 3, 3, 2};
    size = sizeof(arr4)/sizeof(arr4[0]);
    k = 3;
    moreThanNdK(arr4, size, k);

    return 0;
}
```

Output:

```
First Test
Number:4 Count:3

Second Test
Number:2 Count:2

Third Test
Number:2 Count:2

Fourth Test
Number:2 Count:2
Number:3 Count:2
```

Time Complexity: O(nk)
Auxiliary Space: O(k)

Generally asked variations of this problem are, find all elements that appear n/3 times or n/4 times in O(n) time complexity and O(1) extra space.

Hashing can also be an efficient solution. With a good hash function, we can solve the above problem in O(n) time on average. Extra space required hashing would be higher than O(k). Also, hashing cannot be used to solve above variations with O(1) extra space.

Exercise:
The above problem can be solved in O(nLogk) time with the help of more appropriate data structures than array for auxiliary storage of k-1 elements. Suggest a O(nLogk) approach.

This article is contributed by **Kushagra Jaiswal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/given-an-array-of-of-size-n-finds-all-the-elements-that-appear-more-than-nk-times/

Chapter 121

Grouping Countries

Grouping Countries - GeeksforGeeks

People in a group, are sitting in a row numbered from 1 to n. Every has been asked the same question, "How many people of your country are there in the group?"
The answers provided by the people may be incorrect. People of the same country always sit together. If all answers are correct determine the number of distinct countries else print "Invalid Answer".

Examples:

```
Input : ans[] = {1, 3, 2, 2}
Output : Invalid Answer
The second person says there are 3
people from his country however
the person sitting next to him says
there are 2 people. Hence this is
an invalid answer.

Input : ans[] = {1, 1, 2, 2, 4, 4,
                            4, 4}
Output : 4
There are 1 person each representing
two distinct countries. In the next
one there are two people and in the
fourth one there are 4 people from
the same country.
```

Source : ThoughtWorks Application Qualifier Test

This is a basic problem that can be solved in linear time. We will take a variable curr_size that tells us the size of the current country being considered. Whatever the size is, next

'size' number of people should give the same answer in order for a valid group to be formed. If anyone gives a different answer or there are less that that number of people available then the answer is Invalid. Below is the implementation of idea :

C++

```
 // CPP program to count no of distinct
// countries from a given group of people
#include <iostream>
using namespace std;

void countCountries(int ans[], int N)
{
    int total_countries = 0, i = 0;
    bool invalid = false;

    while (i < N) {
        int curr_size = ans[i];

        // Answer is valid if adjacent sitting
        // num people give same answer
        int num = ans[i];
        while (num > 0) {

            // someone gives different answer
            if (ans[i] != curr_size) {
                cout << "Invalid Answer\n";
                return;
            }
            else
                num--;

            // check next person
            i++;
        }

        // one valid country group has
        // been found
        total_countries++;
    }

    cout << "There are " << total_countries
         << " distinct companies in the group.\n";
}

// driver program to test above function
int main()
{
```

```
    int ans[] = { 1, 1, 2, 2, 4, 4, 4, 4 };
    int n = sizeof(ans) / sizeof(ans[0]);
    countCountries(ans, n);
    return 0;
}
```

Java

```
 // Java program to count no of distinct
// countries from a given group of people

class Country
{
    public static void countCountries(int ans[],
                                      int N)
    {
        int total_countries = 0, i = 0;
        boolean invalid = false;

        while (i < N) {
            int curr_size = ans[i];

            // Answer is valid if adjacent sitting
            // num people give same answer
            int num = ans[i];
            while (num > 0) {

            // someone gives different answer
            if (ans[i] != curr_size) {
                System.out.print( "Invalid Answer\n" );
                return;
            }
            else
                num--;

            // check next person
            i++;
        }

        // one valid country group has
        // been found
        total_countries++;
        }

        System.out.print( "There are " + total_countries +
            " distinct companies in the group.\n" );
    }
```

```
    // driver code
    public static void main(String[] args)
    {
        int ans[] = { 1, 1, 2, 2, 4, 4, 4, 4 };
        int n = 8;
        countCountries(ans, n);
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
 # Python3 program to count no of distinct
# countries from a given group of people

def countCountries(ans, N):
    total_countries = 0
    i = 0
    invalid = 0

    while (i < N) :
        curr_size = ans[i]

        # Answer is valid if adjacent sitting
        # num people give same answer
        num = ans[i]
        while (num > 0) :

            # someone gives different answer
            if (ans[i] != curr_size) :
                print("Invalid Answer")
                return;
            else:
                num = num - 1

            # check next person
            i = i + 1

        # one valid country group has
        # been found
        total_countries = total_countries + 1;

    print ("There are ", total_countries,
          " distinct companies in the group.")

# Driven code
ans = [ 1, 1, 2, 2, 4, 4, 4, 4 ];
```

```
n = len(ans);
countCountries(ans, n);

# This code is contributed by "rishabh_jain".
```

C#

```
 // C# program to count no. of distinct
// countries from a given group of people
using System;

class Country {

    // function to count no. of distinct
    // countries from a given group of people
    public static void countCountries(int []ans,
                                      int N)
    {
        int total_countries = 0, i = 0;


        while (i < N) {
            int curr_size = ans[i];

            // Answer is valid if adjacent sitting
            // num people give same answer
            int num = ans[i];
            while (num > 0) {

            // someone gives different answer
            if (ans[i] != curr_size) {
                Console.Write( "Invalid Answer\n" );
                return;
            }
            else
                num--;

            // check next person
            i++;
        }

        // one valid country group
        // has been found
        total_countries++;
        }

        Console.Write("There are " + total_countries +
                      " distinct companies in the group.\n" );
```

```
    }

    // Driver Code
    public static void Main()
    {
        int []ans = { 1, 1, 2, 2, 4, 4, 4, 4 };
        int n = 8;
        countCountries(ans, n);
    }
}

// This code is contributed by nitin mittal
```

PHP

```
 <?php
// PHP program to count no of distinct
// countries from a given group of people

function countCountries($ans, $N)
{
    $total_countries = 0;
    $i = 0;
    $invalid = false;

    while ($i < $N)
    {
        $curr_size = $ans[$i];

        // Answer is valid if adjacent sitting
        // num people give same answer
        $num = $ans[$i];
        while ($num > 0)
        {

            // someone gives different
            // answer
            if ($ans[$i] != $curr_size)
            {
                echo"Invalid Answer\n";
                return;
            }
            else
                $num--;

            // check next person
            $i++;
        }
```

```
        // one valid country group has
        // been found
        $total_countries++;
    }

    echo "There are " , $total_countries
        , " distinct companies in the group.\n";
}

    // Driver Code
    $ans = array(1, 1, 2, 2, 4, 4, 4, 4 );
    $n = sizeof($ans);
    countCountries($ans, $n);

// This code is contributed by nitin mittal.
?>
```

Output:

```
There are 4 distinct companies in the group.
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/grouping-countries/

Chapter 122

Hashtables Chaining with Doubly Linked Lists

Hashtables Chaining with Doubly Linked Lists - GeeksforGeeks

Prerequisite – Hashing Introduction, Hashtable using Singly Linked List & Implementing our Own Hash Table with Separate Chaining in Java

Implementing hash table using Chaining through Doubly Linked List is similar to implementingHashtable using Singly Linked List. The only difference is that every node of Linked List has the address of both, the next and the previous node. This will speed up the process of adding and removing elements from the list, hence the time complexity will be reduced drastically.
Example:

> If we have a Singly linked list:
>
> ```
> 1->2->3->4
> ```
>
> If we are at 3 and there is a need to remove it, then 2 need to be linked with 4 and as from 3, 2 can't be accessed as it is singly linked list. So, the list has to be traversed again i.e O(n), but if we have doubly linked list i.e.
>
> ```
> 1234
> ```
>
> 2 & 4 can be accessed from 3, hence in O(1), 3 can be removed.

Below is the implementation of the above approach:

```
 // C++ implementation of Hashtable
// using doubly linked list
#include <bits/stdc++.h>
using namespace std;

const int tablesize = 25;

// declaration of node
struct hash_node {
    int val, key;
    hash_node* next;
    hash_node* prev;
};

// hashmap's declaration
class HashMap {
public:
    hash_node **hashtable, **top;

    // constructor
    HashMap()
    {
        // create a empty hashtable
        hashtable = new hash_node*[tablesize];
        top = new hash_node*[tablesize];
        for (int i = 0; i < tablesize; i++) {
            hashtable[i] = NULL;
            top[i] = NULL;
        }
    }

    // destructor
    ~HashMap()
    {
        delete[] hashtable;
    }

    // hash function definition
    int HashFunc(int key)
    {
        return key % tablesize;
    }

    // searching method
    void find(int key)
    {
        // Applying hashFunc to find
        // index for given key
```

```
    int hash_val = HashFunc(key);
    bool flag = false;
    hash_node* entry = hashtable[hash_val];

    // if hashtable at that index has some
    // values stored
    if (entry != NULL) {
        while (entry != NULL) {
            if (entry->key == key) {
                flag = true;
            }
            if (flag) {
                cout << "Element found at key "
                     << key << ": ";
                cout << entry->val << endl;
            }
            entry = entry->next;
        }
    }
    if (!flag)
        cout << "No Element found at key "
             << key << endl;
}

// removing an element
void remove(int key)
{
    // Applying hashFunc to find
    // index for given key
    int hash_val = HashFunc(key);
    hash_node* entry = hashtable[hash_val];
    if (entry->key != key || entry == NULL) {
        cout << "Couldn't find any element at this key "
             << key << endl;
        return;
    }

    // if some values are present at that key &
    // traversing the list and removing all values
    while (entry != NULL) {
        if (entry->next == NULL) {
            if (entry->prev == NULL) {
                hashtable[hash_val] = NULL;
                top[hash_val] = NULL;
                delete entry;
                break;
            }
            else {
```

```
                top[hash_val] = entry->prev;
                top[hash_val]->next = NULL;
                delete entry;
                entry = top[hash_val];
            }
        }
        entry = entry->next;
    }
    cout << "Element was successfully removed at the key "
         << key << endl;
}

// inserting method
void add(int key, int value)
{
    // Applying hashFunc to find
    // index for given key
    int hash_val = HashFunc(key);
    hash_node* entry = hashtable[hash_val];

    // if key has no value stored
    if (entry == NULL) {
        // creating new node
        entry = new hash_node;
        entry->val = value;
        entry->key = key;
        entry->next = NULL;
        entry->prev = NULL;
        hashtable[hash_val] = entry;
        top[hash_val] = entry;
    }

    // if some values are present
    else {
        // traversing till the end of
        // the list
        while (entry != NULL)
            entry = entry->next;

        // creating the new node
        entry = new hash_node;
        entry->val = value;
        entry->key = key;
        entry->next = NULL;
        entry->prev = top[hash_val];
        top[hash_val]->next = entry;
        top[hash_val] = entry;
    }
```

```
        cout << "Value " << value << " was successfully"
                " added at key " << key << endl;
    }
};

// Driver Code
int main()
{
    HashMap hash;
    hash.add(4, 5);
    hash.find(4);
    hash.remove(4);
    return 0;
}
```

Output:

```
Value 5 was successfully added at key 4
Element found at key 4: 5
Element was successfully removed at the key 4
```

Source

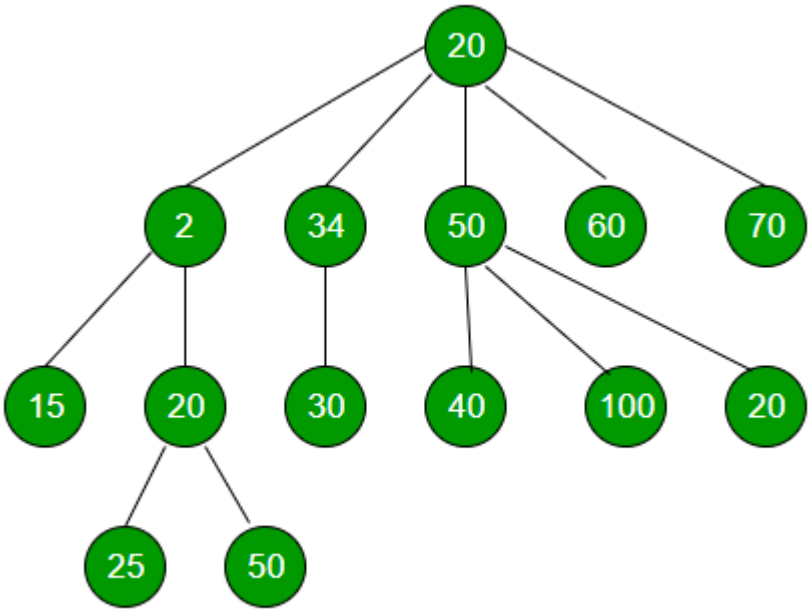
https://www.geeksforgeeks.org/hashtables-chaining-with-doubly-linked-lists/

Chapter 123

Immediate Smaller element in an N-ary Tree

Immediate Smaller element in an N-ary Tree - GeeksforGeeks

Given an element x, task is to find the value of its immediate smaller element.



Example :

```
Input : x = 30 (for above tree)
Output : Immediate smaller element is 25
```

Explanation : Elements 2, 15, 20 and 25 are smaller than x i.e, 30, but 25 is the immediate smaller element and hence the answer.
Approach :

- Let ***res*** be the resultant node.
- Initialize the resultant Node as NULL.
- For every Node, check if data of root is greater than res, but less than x. if yes, update res.
- Recursively do the same for all nodes of the given Generic Tree.
- Return res, and res->key would be the immediate smaller element.

Below is the implementation of above approach :

```
 // C++ program to find immediate Smaller
// Element of a given element in a n-ary tree.
#include <bits/stdc++.h>
using namespace std;

// class of a node of an n-ary tree
class Node {

public:
    int key;
    vector<Node*> child;

    // constructor
    Node(int data)
    {
        key = data;
    }
};

// Function to find immediate Smaller Element
// of a given number x
void immediateSmallerElementUtil(Node* root,
                             int x, Node** res)
{
    if (root == NULL)
        return;

    // if root is greater than res, but less
    // than x, then update res
    if (root->key < x)
        if (!(*res) || (*res)->key < root->key)
```

```
            *res = root; // Updating res

    // Number of children of root
    int numChildren = root->child.size();

    // Recursive calling for every child
    for (int i = 0; i < numChildren; i++)
        immediateSmallerElementUtil(root->child[i], x, res);

    return;
}

// Function to return immediate Smaller
// Element of x in tree
Node* immediateSmallerElement(Node* root, int x)
{
    // resultant node
    Node* res = NULL;

    // calling helper function and using
    // pass by reference
    immediateSmallerElementUtil(root, x, &res);

    return res;
}

// Driver program
int main()
{
    // Creating a generic tree
    Node* root = new Node(20);
    (root->child).push_back(new Node(2));
    (root->child).push_back(new Node(34));
    (root->child).push_back(new Node(50));
    (root->child).push_back(new Node(60));
    (root->child).push_back(new Node(70));
    (root->child[0]->child).push_back(new Node(15));
    (root->child[0]->child).push_back(new Node(20));
    (root->child[1]->child).push_back(new Node(30));
    (root->child[2]->child).push_back(new Node(40));
    (root->child[2]->child).push_back(new Node(100));
    (root->child[2]->child).push_back(new Node(20));
    (root->child[0]->child[1]->child).push_back(new Node(25));
    (root->child[0]->child[1]->child).push_back(new Node(50));

    int x = 30;

    cout << "Immediate smaller element of " << x << " is ";
```

```
    cout << immediateSmallerElement(root, x)->key << endl;

    return 0;
}
```

Output :

```
Immediate smaller element of 30 is 25
```

Time Complexity : O(N), where N is the number of nodes in N-ary Tree.
Auxiliary Space : O(N), for recursive call(worst case when a node has N number of childs)

Source

https://www.geeksforgeeks.org/immediate-smaller-element-n-ary-tree/

Chapter 124

In-place replace multiple occurrences of a pattern

In-place replace multiple occurrences of a pattern - GeeksforGeeks

Given a string and a pattern, replace multiple occurrences of a pattern by character 'X'. The conversion should be in-place and solution should replace **multiple consecutive (and non-overlapping) occurrences of a pattern by a single 'X'**.

```
String - GeeksForGeeks
Pattern - Geeks
Output: XforX

String - GeeksGeeks
Pattern - Geeks
Output: X

String - aaaa
Pattern - aa
Output: X

String - aaaaa
Pattern - aa
Output: Xa
```

The idea is to maintain two index i and j for in-place replacement. Index i always points to next character in the output string. Index j traverses the string and searches for one or more pattern match. If a match is found, we put character 'X' at index i and increment index i by 1 and index j by length of the pattern. Index i is increment only once if we find multiple consecutive occurrences of the pattern. If the pattern is not found, we copy current character at index j to index i and increment both i and j by 1. Since pattern length

is always more than equal to 1 and replacement is only 1 character long, we would never overwrite unprocessed characters i.e j >= i is invariant.

```
 // C++ program to in-place replace multiple
// occurrences of a pattern by character 'X'
#include <bits/stdc++.h>
using namespace std;

// returns true if pattern is prefix of str
bool compare(char* str, char* pattern)
{
    for (int i = 0; pattern[i]; i++)
        if (str[i] != pattern[i])
            return false;
    return true;
}

// Function to in-place replace multiple
// occurrences of a pattern by character 'X'
void replacePattern(char *str, char* pattern)
{
    // If pattern is null or empty string,
    // nothing needs to be done
    if (pattern == NULL)
        return;

    int len = strlen(pattern);
    if (len == 0)
       return;

    int i = 0, j = 0;
    int count;

    // for each character
    while (str[j])
    {
        count = 0;

        // compare str[j..j+len] with pattern
        while (compare(str+j, pattern))
        {
            // increment j by length of pattern
            j = j + len;
            count++;
        }

        // If single or multiple occurrences of pattern
        // is found, replace it by character 'X'
```

```
        if (count > 0)
            str[i++] = 'X';

        // copy character at current position j
        // to position i and increment i and j
        if (str[j])
            str[i++] = str[j++];
    }

    // add a null character to terminate string
    str[i] = '\0';
}

// Driver code
int main()
{
    char str[] = "GeeksforGeeks";
    char pattern[] = "Geeks";

    replacePattern(str, pattern);
    cout << str;

    return 0;
}
```

Output :

```
XforX
```

The time complexity of above algorithm is O(n*m), where n is length of string and m is length of the pattern.

Source

https://www.geeksforgeeks.org/place-replace-multiple-occurrences-pattern/

Chapter 125

Interpolation Search

Interpolation Search - GeeksforGeeks

Given a sorted array of n uniformly distributed values arr[], write a function to search for a particular element x in the array.

Linear Search finds the element in O(n) time, Jump Search takes O(√ n) time and Binary Search take O(Log n) time.
The Interpolation Search is an improvement over Binary Search for instances, where the values in a sorted array are uniformly distributed. Binary Search always goes to the middle element to check. On the other hand, interpolation search may go to different locations according to the value of the key being searched. For example, if the value of the key is closer to the last element, interpolation search is likely to start search toward the end side.

To find the position to be searched, it uses following formula.

```
// The idea of formula is to return higher value of pos
// when element to be searched is closer to arr[hi]. And
// smaller value when closer to arr[lo]
pos = lo + [ (x-arr[lo])*(hi-lo) / (arr[hi]-arr[Lo]) ]

arr[] ==> Array where elements need to be searched
x     ==> Element to be searched
lo    ==> Starting index in arr[]
hi    ==> Ending index in arr[]
```

Algorithm
Rest of the Interpolation algorithm is same except the above partition logic.

Step1: In a loop, calculate the value of "pos" using the probe position formula.
Step2: If it is a match, return the index of the item, and exit.
Step3: If the item is less than arr[pos], calculate the probe position of the left sub-array. Otherwise calculate the same in the right sub-array.
Step4: Repeat until a match is found or the sub-array reduces to zero.

Below is the implementation of algorithm.

C

```
 // C program to implement interpolation search
#include<stdio.h>

// If x is present in arr[0..n-1], then returns
// index of it, else returns -1.
int interpolationSearch(int arr[], int n, int x)
{
    // Find indexes of two corners
    int lo = 0, hi = (n - 1);

    // Since array is sorted, an element present
    // in array must be in range defined by corner
    while (lo <= hi && x >= arr[lo] && x <= arr[hi])
    {
        // Probing the position with keeping
        // uniform distribution in mind.
        int pos = lo + (((double)(hi-lo) /
              (arr[hi]-arr[lo]))*(x - arr[lo]));

        // Condition of target found
        if (arr[pos] == x)
            return pos;

        // If x is larger, x is in upper part
        if (arr[pos] < x)
            lo = pos + 1;

        // If x is smaller, x is in the lower part
        else
            hi = pos - 1;
    }
    return -1;
}

// Driver Code
int main()
{
    // Array of items on which search will
    // be conducted.
    int arr[] =  {10, 12, 13, 16, 18, 19, 20, 21, 22, 23,
                  24, 33, 35, 42, 47};
    int n = sizeof(arr)/sizeof(arr[0]);

    int x = 18; // Element to be searched
```

```
    int index = interpolationSearch(arr, n, x);

    // If element was found
    if (index != -1)
        printf("Element found at index %d", index);
    else
        printf("Element not found.");
    return 0;
}
```

Java

```
 // Java program to implement interpolation search

class Test
{
    // Array of items on which search will
    // be conducted.
    static int arr[] = new int[]{10, 12, 13, 16, 18, 19, 20, 21, 22, 23,
                                        24, 33, 35, 42, 47};

    // If x is present in arr[0..n-1], then returns
    // index of it, else returns -1.
    static int interpolationSearch(int x)
    {
        // Find indexes of two corners
        int lo = 0, hi = (arr.length - 1);

        // Since array is sorted, an element present
        // in array must be in range defined by corner
        while (lo <= hi && x >= arr[lo] && x <= arr[hi])
        {
            // Probing the position with keeping
            // uniform distribution in mind.
            int pos = lo + (((hi-lo) /
                  (arr[hi]-arr[lo]))*(x - arr[lo]));

            // Condition of target found
            if (arr[pos] == x)
                return pos;

            // If x is larger, x is in upper part
            if (arr[pos] < x)
                lo = pos + 1;

            // If x is smaller, x is in the lower part
            else
                hi = pos - 1;
```

```
        }
        return -1;
    }

    // Driver method
    public static void main(String[] args)
    {
         int x = 18; // Element to be searched
         int index = interpolationSearch(x);

         // If element was found
         if (index != -1)
               System.out.println("Element found at index " + index);
            else
               System.out.println("Element not found.");
    }
}
```

Python

```
 # Python program to implement interpolation search

# If x is present in arr[0..n-1], then returns
# index of it, else returns -1
def interpolationSearch(arr, n, x):
    # Find indexs of two corners
    lo = 0
    hi = (n - 1)

    # Since array is sorted, an element present
    # in array must be in range defined by corner
    while lo <= hi and x >= arr[lo] and x <= arr[hi]:
        # Probing the position with keeping
        # uniform distribution in mind.
        pos  = lo + int(((float(hi - lo) /
            ( arr[hi] - arr[lo])) * ( x - arr[lo])))

        # Condition of target found
        if arr[pos] == x:
            return pos

        # If x is larger, x is in upper part
        if arr[pos] < x:
            lo = pos + 1;

        # If x is smaller, x is in lower part
        else:
            hi = pos - 1;
```

```
    return -1

# Driver Code
# Array of items oin which search will be conducted
arr = [10, 12, 13, 16, 18, 19, 20, 21, \
                22, 23, 24, 33, 35, 42, 47]
n = len(arr)

x = 18 # Element to be searched
index = interpolationSearch(arr, n, x)

if index != -1:
    print "Element found at index",index
else:
    print "Element not found"

# This code is contributed by Harshit Agrawal
```

C#

```
 // C# program to implement
// interpolation search
using System;

class GFG
{
    // Array of items on which
    // search will be conducted.
    static int []arr = new int[]{10, 12, 13, 16, 18,
                                 19, 20, 21, 22, 23,
                                 24, 33, 35, 42, 47};

    // If x is present in
    // arr[0..n-1], then
    // returns index of it,
    // else returns -1.
    static int interpolationSearch(int x)
    {
        // Find indexes of
        // two corners
        int lo = 0, hi = (arr.Length - 1);

        // Since array is sorted,
        // an element present in
        // array must be in range
        // defined by corner
        while (lo <= hi &&
```

```
                x >= arr[lo] &&
                x <= arr[hi])
        {
            // Probing the position
            // with keeping uniform
            // distribution in mind.
            int pos = lo + (((hi - lo) /
                            (arr[hi] - arr[lo])) *
                                  (x - arr[lo]));

            // Condition of
            // target found
            if (arr[pos] == x)
                return pos;

            // If x is larger, x
            // is in upper part
            if (arr[pos] < x)
                lo = pos + 1;

            // If x is smaller, x
            // is in the lower part
            else
                hi = pos - 1;
        }
        return -1;
    }

    // Driver Code
    public static void Main()
    {
        int x = 18; // Element to be searched
        int index = interpolationSearch(x);

        // If element was found
        if (index != -1)
            Console.WriteLine("Element found " +
                                    "at index " +
                                          index);
            else
            Console.WriteLine("Element not found.");
    }
}

// This code is contributed by anuj_67.
```

Output:

```
Element found at index 4
```

Time Complexity: If elements are uniformly distributed, then **O (log log n))**. In worst case it can take upto O(n).
Auxiliary Space: O(1)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/interpolation-search/

Chapter 126

Interpolation search vs Binary search

Interpolation search vs Binary search - GeeksforGeeks

Interpolation search works better than Binary Search for a sorted and uniformly distributed array.

On average the interpolation search makes about log(log(n)) comparisons (if the elements are uniformly distributed), where n is the number of elements to be searched. In the worst case (for instance where the numerical values of the keys increase exponentially) it can make up to O(n) comparisons.

Interpolation Search Article

Sources:
http://en.wikipedia.org/wiki/Interpolation_search

Source

https://www.geeksforgeeks.org/g-fact-84/

Chapter 127

Josephus Problem | (Iterative Solution)

Josephus Problem | (Iterative Solution) - GeeksforGeeks

There are N Children are seated on N chairs arranged around a circle. The chairs are numbered from 1 to N. The game starts going in circles counting the children starting with the first chair. Once the count reaches K, that child leaves the game, removing his/her chair. The game starts again, beginning with the next chair in the circle. The last child remaining in the circle is the winner. Find the child that wins the game.

Examples:

```
Input : N = 5, K = 2
Output : 3
Firstly, the child at position 2 is out,
then position 4 goes out, then position 1
Finally, the child at position 5 is out.
So the position 3 survives.

Input : 7 4
Output : 2
```

We have discussed a recursive solution for Josephus Problem. The given solution is better than the recursive solution of Josephus Solution which is not suitable for large inputs as it gives stack overflow. The time complexity is O(N).

Approach – In the algorithm, we use sum variable to find out the chair to be removed. The current chair position is calculated by adding the chair count K to the previous position i.e. sum and modulus of the sum. At last we return sum+1 as numbering starts from 1 to N.

```
// Iterative solution for Josephus Problem
```

```
#include <bits/stdc++.h>
using namespace std;

// Function for finding the winning child.
long long int find(long long int n, long long int k)
{
    long long int sum = 0, i;

    // For finding out the removed
    // chairs in each iteration
    for (i = 2; i <= n; i++)
        sum = (sum + k) % i;

    return sum + 1;
}

// Driver function to find the winning child
int main()
{
    int n = 14, k = 2;
    cout << find(n, k);
    return 0;
}
```

Output:

```
13
```

Source

https://www.geeksforgeeks.org/josephus-problem-iterative-solution/

Chapter 128

Jump Search

Jump Search - GeeksforGeeks

Like Binary Search, Jump Search is a searching algorithm for sorted arrays. The basic idea is to check fewer elements (than linear search) by jumping ahead by fixed steps or skipping some elements in place of searching all elements.

For example, suppose we have an array arr[] of size n and block (to be jumped) size m. Then we search at the indexes arr[0], arr[m], arr[2m]…..arr[km] and so on. Once we find the interval (arr[km] < x < arr[(k+1)m]), we perform a linear search operation from the index km to find the element x.

Let's consider the following array: (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610). Length of the array is 16. Jump search will find the value of 55 with the following steps assuming that the block size to be jumped is 4.
STEP 1: Jump from index 0 to index 4;
STEP 2: Jump from index 4 to index 8;
STEP 3: Jump from index 8 to index 16;
STEP 4: Since the element at index 16 is greater than 55 we will jump back a step to come to index 9.
STEP 5: Perform linear search from index 9 to get the element 55.

What is the optimal block size to be skipped?
In the worst case, we have to do n/m jumps and if the last checked value is greater than the element to be searched for, we perform m-1 comparisons more for linear search. Therefore the total number of comparisons in the worst case will be ((n/m) + m-1). The value of the function ((n/m) + m-1) will be minimum when m = √n. Therefore, the best step size is m = **√n.**

C++

```
 // C++ program to implement Jump Search

#include <bits/stdc++.h>
using namespace std;
```

```
int jumpSearch(int arr[], int x, int n)
{
    // Finding block size to be jumped
    int step = sqrt(n);

    // Finding the block where element is
    // present (if it is present)
    int prev = 0;
    while (arr[min(step, n)-1] < x)
    {
        prev = step;
        step += sqrt(n);
        if (prev >= n)
            return -1;
    }

    // Doing a linear search for x in block
    // beginning with prev.
    while (arr[prev] < x)
    {
        prev++;

        // If we reached next block or end of
        // array, element is not present.
        if (prev == min(step, n))
            return -1;
    }
    // If element is found
    if (arr[prev] == x)
        return prev;

    return -1;
}

// Driver program to test function
int main()
{
    int arr[] = { 0, 1, 1, 2, 3, 5, 8, 13, 21,
                34, 55, 89, 144, 233, 377, 610 };
    int x = 55;
    int n = sizeof(arr) / sizeof(arr[0]);

    // Find the index of 'x' using Jump Search
    int index = jumpSearch(arr, x, n);

    // Print the index where 'x' is located
    cout << "\nNumber " << x << " is at index " << index;
```

```
    return 0;
}

// Contributed by nuclode
```

Java

```
 // Java program to implement Jump Search.
public class JumpSearch
{
    public static int jumpSearch(int[] arr, int x)
    {
        int n = arr.length;

        // Finding block size to be jumped
        int step = (int)Math.floor(Math.sqrt(n));

        // Finding the block where element is
        // present (if it is present)
        int prev = 0;
        while (arr[Math.min(step, n)-1] < x)
        {
            prev = step;
            step += (int)Math.floor(Math.sqrt(n));
            if (prev >= n)
                return -1;
        }

        // Doing a linear search for x in block
        // beginning with prev.
        while (arr[prev] < x)
        {
            prev++;

            // If we reached next block or end of
            // array, element is not present.
            if (prev == Math.min(step, n))
                return -1;
        }

        // If element is found
        if (arr[prev] == x)
            return prev;

        return -1;
    }

    // Driver program to test function
```

```
    public static void main(String [ ] args)
    {
        int arr[] = { 0, 1, 1, 2, 3, 5, 8, 13, 21,
                    34, 55, 89, 144, 233, 377, 610};
        int x = 55;

        // Find the index of 'x' using Jump Search
        int index = jumpSearch(arr, x);

        // Print the index where 'x' is located
        System.out.println("\nNumber " + x +
                           " is at index " + index);
    }
}
```

Python3

```
 # Python3 code to implement Jump Search
import math

def jumpSearch( arr , x , n ):

    # Finding block size to be jumped
    step = math.sqrt(n)

    # Finding the block where element is
    # present (if it is present)
    prev = 0
    while arr[int(min(step, n)-1)] < x:
        prev = step
        step += math.sqrt(n)
        if prev >= n:
            return -1

    # Doing a linear search for x in
    # block beginning with prev.
    while arr[int(prev)] < x:
        prev += 1

        # If we reached next block or end
        # of array, element is not present.
        if prev == min(step, n):
            return -1

    # If element is found
    if arr[int(prev)] == x:
        return prev
```

```
    return -1

# Driver code to test function
arr = [ 0, 1, 1, 2, 3, 5, 8, 13, 21,
    34, 55, 89, 144, 233, 377, 610 ]
x = 55
n = len(arr)

# Find the index of 'x' using Jump Search
index = jumpSearch(arr, x, n)

# Print the index where 'x' is located
print("Number" , x, "is at index" ,"%.0f"%index)

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // C# program to implement Jump Search.
using System;
public class JumpSearch
{
    public static int jumpSearch(int[] arr, int x)
    {
        int n = arr.Length;

        // Finding block size to be jumped
        int step = (int)Math.Floor(Math.Sqrt(n));

        // Finding the block where element is
        // present (if it is present)
        int prev = 0;
        while (arr[Math.Min(step, n)-1] < x)
        {
            prev = step;
            step += (int)Math.Floor(Math.Sqrt(n));
            if (prev >= n)
                return -1;
        }

        // Doing a linear search for x in block
        // beginning with prev.
        while (arr[prev] < x)
        {
            prev++;

            // If we reached next block or end of
            // array, element is not present.
```

```
            if (prev == Math.Min(step, n))
                return -1;
        }

        // If element is found
        if (arr[prev] == x)
            return prev;

        return -1;
    }

    // Driver program to test function
    public static void Main()
    {
        int[] arr = { 0, 1, 1, 2, 3, 5, 8, 13, 21,
                    34, 55, 89, 144, 233, 377, 610};
        int x = 55;

        // Find the index of 'x' using Jump Search
        int index = jumpSearch(arr, x);

        // Print the index where 'x' is located
        Console.Write("Number " + x +
                            " is at index " + index);
    }
}
```

PHP

```
 <?php
// PHP program to implement Jump Search

function jumpSearch($arr, $x, $n)
{
    // Finding block size to be jumped
    $step = sqrt($n);

    // Finding the block where element is
    // present (if it is present)
    $prev = 0;
    while ($arr[min($step, $n)-1] < $x)
    {
        $prev = $step;
        $step += sqrt($n);
        if ($prev >= $n)
            return -1;
    }
```

```
    // Doing a linear search for x in block
    // beginning with prev.
    while ($arr[$prev] < $x)
    {
        $prev++;

        // If we reached next block or end of
        // array, element is not present.
        if ($prev == min($step, $n))
            return -1;
    }
    // If element is found
    if ($arr[$prev] == $x)
        return $prev;

    return -1;
}

// Driver program to test function
$arr = array( 0, 1, 1, 2, 3, 5, 8, 13, 21,
                34, 55, 89, 144, 233, 377, 610 );
$x = 55;
$n = sizeof($arr) / sizeof($arr[0]);

// Find the index of '$x' using Jump Search
$index = jumpSearch($arr, $x, $n);

// Print the index where '$x' is located
echo "Number ".$x." is at index " .$index;
return 0;
?>
```

Output:

```
Number 55 is at index 10
```

Time Complexity : O(√n)
Auxiliary Space : O(1)

Important points:

- Works only sorted arrays.
- The optimal size of a block to be jumped is O(√ n). This makes the time complexity of Jump Search O(√ n).
- The time complexity of Jump Search is between Linear Search ((O(n)) and Binary Search (O (Log n)).

- Binary Search is better than Jump Search, but Jump search has an advantage that we traverse back only once (Binary Search may require up to O(Log n) jumps, consider a situation where the element to be search is the smallest element or smaller than the smallest). So in a systems where jumping back is costly, we use Jump Search.

References:
https://en.wikipedia.org/wiki/Jump_search

Improved By : ChitraNayal

Source

https://www.geeksforgeeks.org/jump-search/

Chapter 129

K distant string

K distant string - GeeksforGeeks

Given a string of length n and a non-negative integer k. Find k distant string of given string.

Distance between two letters is difference between their positions in the alphabet. for example:

- dist(c, e) = dist(e, c) = 2.
- dist(a, z) = dist(z, a) = 25.

By using this concept, the distance between two strings is the sum of distances of corresponding letters. For example :

- dist(af, hf) = dist(a, h) + dist(f, f) = 7 + 0 = 7.

Given a string and a distance k. Task is to find a string such that distance of the result string is k from given string. If k distant string is not possible, then print "No".
Note: There may be exist multiple solutions. We meed to find one of them.

Examples :

```
Input : bear
        k = 26
Output : zcar
Here, dist(bear, zcar) =
      dist(b, z) + dist(e, c) +
      + dist(a, a) + dist(r, r)
      = 24 + 2 + 0 + 0
      = 26

Input : af
```

```
        k = 7
Output : hf
Here, dist(af, hf) = dist(a, h) + dist(f, f)
                   = 7 + 0
                   = 7

Input : hey
        k = 1000
Output : No
Explaination :
No such string exists.
```

There is no solution if the given required distance is too big. Think what is the maximum possible distance for the given string. Or the more useful thing — how to construct the lost string to maximize the distance? Treat each letter separately and replace it with the most distant letter. For example, we should replace 'c' with 'z', and we should replace 'y' with 'a'. To be more precise, for first 13 letters of the alphabet the most distant letter is 'z', and for other letters it is 'a'.

The approach is simple, iterate over letters of the given string and greedily change them. A word "greedily" means when changing a letter, don't care about the next letters. Generally, there must be distant letters, because there may not be a solution otherwise. For each letter of the given string change it into the most distant letter, unless the total distance would be too big. As letters are changed, decrease the remaining required distance. So, for each letter of the given string consider only letters not exceeding the remaining distance, and among them choose the most distant one.

CPP and JAVA Implementation:

CPP

```
 // CPP program to find the k distant string
#include <bits/stdc++.h>
using namespace std;

// function to find the
// lost string
string findKDistantString(string str, int k)
{
    int n = str.length();

    for (int i = 0; i < n; ++i) {

        char best_letter = str[i];
        int best_distance = 0;

        for (char maybe = 'a';
             maybe <= 'z'; ++maybe)
        {
```

```
            int distance = abs(maybe - str[i]);

            // check if "distance <= k",
            // so that, the total distance
            // will not exceed among
            // letters with "distance <= k"
            if (distance <= k && distance >
                              best_distance)
            {
                best_distance = distance;
                best_letter = maybe;
            }
        }

        // decrease the remaining
        // distance
        k -= best_distance;
        str[i] = best_letter;

    }

    assert(k >= 0);
    // we found a correct
    // string only if "k == 0"
    if (k > 0)
        return "No";
    else
        return str;
}

// driver function
int main()
{
    string str = "bear";
    int k = 26;
    cout << findKDistantString(str, k) << endl;

    str = "af";
    k = 7;

    cout << findKDistantString(str, k) << endl;
    return 0;
}
```

Java

```
 // Java program to find k distant string
import java.util.*;
```

```
import java.lang.*;

public class GfG {

    // function to find
    // the lost string
    public static String findKDistantString
                        (String str1, int k)
    {
        int n = str1.length();
        char[] str = str1.toCharArray();

        for (int i = 0; i < n; ++i) {
            char best_letter = str[i];
            int best_distance = 0;

            for (char maybe = 'a';
                 maybe <= 'z'; ++maybe)
            {
                int distance =
                    Math.abs(maybe - str[i]);

                // Check if "distance <= k"
                // so that it should not
                // exceed the total distance
                // among letters with "distance
                // <= k" we choose the most
                // distant one
                if (distance <= k && distance
                                > best_distance)
                {
                    best_distance = distance;
                    best_letter = maybe;
                }
            }

            // we decrease the remaining
            // distance
            k -= best_distance;
            str[i] = best_letter;
        }

        assert(k >= 0);

        // Correct string only
        // if "k == 0"
        if (k > 0)
            return "No";
```

```
        else
            return (new String(str));
    }
    public static void main(String argc[])
    {
        String str = "bear";
        int k = 26;
        System.out.println(findKDistantString(str, k));

        str = "af";
        k = 7;
        System.out.println(findKDistantString(str, k));
    }
}
```

Python3

```
 # Python implementation to check if
# both halves of the string have
# at least one different character

MAX = 26

# Function which break string into two halves
# Counts frequency of characters in each half
# Compares the two counter array and returns
# true if these counter arrays differ
def function(st):
    global MAX
    l = len(st)

    # Declaration and initialization
    # of counter array
    counter1, counter2 = [0]*MAX, [0]*MAX

    for i in range(l//2):
        counter1[ord(st[i]) - ord('a')] += 1

    for i in range(l//2, l):
        counter2[ord(st[i]) - ord('a')] += 1

    for i in range(MAX):
        if (counter2[i] != counter1[i]):
            return True

    return False
```

```
# Driver function
st = "abcasdsabcae"
if function(st): print("Yes, both halves differ by at least one character")
else: print("No, both halves do not differ at all")

# This code is contributed by Ansu Kumari
```

C#

```
 // C# program to find k distant string
using System;

class GfG {

    // function to find the lost string
    public static String findKDistantString
                        (string str1, int k)
    {
        int n = str1.Length;
        char []str = str1.ToCharArray();

        for (int i = 0; i < n; ++i) {
            char best_letter = str[i];
            int best_distance = 0;

            for (char maybe = 'a';
                maybe <= 'z'; ++maybe)
            {
                int distance =
                    Math.Abs(maybe - str[i]);

                // Check if "distance <= k"
                // so that it should not
                // exceed the total distance
                // among letters with "distance
                // <= k" we choose the most
                // distant one
                if (distance <= k && distance
                                > best_distance)
                {
                    best_distance = distance;
                    best_letter = maybe;
                }
            }

            // we decrease the remaining
            // distance
            k -= best_distance;
```

```
            str[i] = best_letter;
        }

        //(k >= 0);

        // Correct string only
        // if "k == 0"
        if (k > 0)
            return "No";
        else
            return (new string(str));
    }

    // Driver code
    public static void Main()
    {
        string str = "bear";
        int k = 26;
        Console.WriteLine(
              findKDistantString(str, k));

        str = "af";
        k = 7;
        Console.Write(
              findKDistantString(str, k));
    }
}

// This code is contributed by Nitin millal.
```

Output:

```
zcar
hf
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/find-k-distant-string/

Chapter 130

Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1

Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1 - GeeksforGeeks

Given an n x n matrix, where every row and column is sorted in non-decreasing order. Find the kth smallest element in the given 2D array.

For example, consider the following 2D array.

```
        10, 20, 30, 40
        15, 25, 35, 45
        24, 29, 37, 48
        32, 33, 39, 50
The 3rd smallest element is 20 and 7th smallest element is 30
```

The idea is to use min heap. Following are detailed step.
1) Build a min heap of elements from first row. A heap entry also stores row number and column number.
2) Do following k times.
…a) Get minimum element (or root) from min heap.
…b) Find row number and column number of the minimum element.
…c) Replace root with the next element from same column and min-heapify the root.
3) Return the last extracted root.

Following is C++ implementation of above algorithm.

```
 // kth largest element in a 2d array sorted row-wise and column-wise
#include<iostream>
```

```
#include<climits>
using namespace std;

// A structure to store an entry of heap.  The entry contains
// a value from 2D array, row and column numbers of the value
struct HeapNode {
    int val;  // value to be stored
    int r;    // Row number of value in 2D array
    int c;    // Column number of value in 2D array
};

// A utility function to swap two HeapNode items.
void swap(HeapNode *x, HeapNode *y) {
    HeapNode z = *x;
    *x = *y;
    *y = z;
}

// A utility function to minheapify the node harr[i] of a heap
// stored in harr[]
void minHeapify(HeapNode harr[], int i, int heap_size)
{
    int l = i*2 + 1;
    int r = i*2 + 2;
    int smallest = i;
    if (l < heap_size && harr[l].val < harr[i].val)
        smallest = l;
    if (r < heap_size && harr[r].val < harr[smallest].val)
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        minHeapify(harr, smallest, heap_size);
    }
}

// A utility function to convert harr[] to a max heap
void buildHeap(HeapNode harr[], int n)
{
    int i = (n - 1)/2;
    while (i >= 0)
    {
        minHeapify(harr, i, n);
        i--;
    }
}

// This function returns kth smallest element in a 2D array mat[][]
```

```
int kthSmallest(int mat[4][4], int n, int k)
{
    // k must be greater than 0 and smaller than n*n
    if (k <= 0 || k > n*n)
       return INT_MAX;

    // Create a min heap of elements from first row of 2D array
    HeapNode harr[n];
    for (int i = 0; i < n; i++)
        harr[i] =  {mat[0][i], 0, i};
    buildHeap(harr, n);

    HeapNode hr;
    for (int i = 0; i < k; i++)
    {
       // Get current heap root
       hr = harr[0];

       // Get next value from column of root's value. If the
       // value stored at root was last value in its column,
       // then assign INFINITE as next value
       int nextval = (hr.r < (n-1))? mat[hr.r + 1][hr.c]: INT_MAX;

       // Update heap root with next value
       harr[0] =  {nextval, (hr.r) + 1, hr.c};

       // Heapify root
       minHeapify(harr, 0, n);
    }

    // Return the value at last extracted root
    return hr.val;
}

// driver program to test above function
int main()
{
  int mat[4][4] = { {10, 20, 30, 40},
                    {15, 25, 35, 45},
                    {25, 29, 37, 48},
                    {32, 33, 39, 50},
                  };
  cout << "7th smallest element is " << kthSmallest(mat, 4, 7);
  return 0;
}
```

Output:

```
7th smallest element is 30
```

Time Complexity: The above solution involves following steps.
1) Build a min heap which takes O(n) time
2) Heapify k times which takes O(kLogn) time.
Therefore, overall time complexity is O(n + kLogn) time.

The above code can be optimized to build a heap of size k when k is smaller than n. In that case, the kth smallest element must be in first k rows and k columns.

We will soon be publishing more efficient algorithms for finding the kth smallest element.

This article is compiled by Ravi Gupta. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/kth-smallest-element-in-a-row-wise-and-column-wise-sorted-2d-array-set-1/

Chapter 131

K'th Smallest/Largest Element in Unsorted Array | Set 1

K'th Smallest/Largest Element in Unsorted Array | Set 1 - GeeksforGeeks

Given an array and a number k where k is smaller than size of array, we need to find the k'th smallest element in the given array. It is given that ll array elements are distinct.

Examples:

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 3
Output: 7

Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 4
Output: 10
```

We have discussed a similar problem to print k largest elements.

Method 1 (Simple Solution)
A Simple Solution is to sort the given array using a O(nlogn) sorting algorithm like Merge Sort, Heap Sort, etc and return the element at index k-1 in the sorted array. Time Complexity of this solution is O(nLogn).

C++

```
 // Simple C++ program to find k'th smallest element
#include<iostream>
#include<algorithm>
using namespace std;
```

```
// Function to return k'th smallest element in a given array
int kthSmallest(int arr[], int n, int k)
{
    // Sort the given array
    sort(arr, arr+n);

    // Return k'th element in the sorted array
    return arr[k-1];
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 19};
    int n = sizeof(arr)/sizeof(arr[0]), k = 2;
    cout << "K'th smallest element is " <<  kthSmallest(arr, n, k);
    return 0;
}
```

Java

```
 // Java code for kth smallest element
// in an array
import java.util.Arrays;
import java.util.Collections;

class GFG
{
    // Function to return k'th smallest
    // element in a given array
    public static int kthSmallest(Integer [] arr,
                                          int k)
    {
        // Sort the given array
        Arrays.sort(arr);

        // Return k'th element in
        // the sorted array
        return arr[k-1];
    }

    // driver program
    public static void main(String[] args)
    {
        Integer arr[] = new Integer[]{12, 3, 5, 7, 19};
        int k = 2;
        System.out.print( "K'th smallest element is "+
                           kthSmallest(arr, k) );
```

```
    }
}

// This code is contributed by Chhavi
```

C#

```
 // C# code for kth smallest element
// in an array
using System;

class GFG {

    // Function to return k'th smallest
    // element in a given array
    public static int kthSmallest(int []arr,
                                  int k)
    {

        // Sort the given array
        Array.Sort(arr);

        // Return k'th element in
        // the sorted array
        return arr[k-1];
    }

    // driver program
    public static void Main()
    {
        int []arr = new int[]{12, 3, 5,
                              7, 19};
        int k = 2;
        Console.Write( "K'th smallest element"
              + " is "+ kthSmallest(arr, k) );
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// Simple PHP program to find
// k'th smallest element

// Function to return k'th smallest
```

```
// element in a given array
function kthSmallest($arr, $n, $k)
{

    // Sort the given array
    sort($arr);

    // Return k'th element
    // in the sorted array
    return $arr[$k - 1];
}

    // Driver Code
    $arr = array(12, 3, 5, 7, 19);
    $n =count($arr);
    $k = 2;
    echo "K'th smallest element is "
       , kthSmallest($arr, $n, $k);

// This code is contributed by anuj_67.
?>


K'th smallest element is 5
```

Method 2 (Using Min Heap – HeapSelect)
We can find k'th smallest element in time complexity better than O(nLogn). A simple optomization is to create a Min Heapof the given n elements and call extractMin() k times. The following is C++ implementation of above method.

```
 // A C++ program to find k'th smallest element using min heap
#include<iostream>
#include<climits>
using namespace std;

// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Min Heap
class MinHeap
{
    int *harr; // pointer to array of elements in heap
    int capacity; // maximum possible size of min heap
    int heap_size; // Current number of elements in min heap
public:
    MinHeap(int a[], int size); // Constructor
    void MinHeapify(int i);  //To minheapify subtree rooted with index i
```

```
    int parent(int i) { return (i-1)/2; }
    int left(int i) { return (2*i + 1); }
    int right(int i) { return (2*i + 2); }

    int extractMin();  // extracts root (minimum) element
    int getMin() { return harr[0]; } // Returns minimum
};

MinHeap::MinHeap(int a[], int size)
{
    heap_size = size;
    harr = a;  // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// Method to remove minimum element (or root) from min heap
int MinHeap::extractMin()
{
    if (heap_size == 0)
        return INT_MAX;

    // Store the minimum vakue.
    int root = harr[0];

    // If there are more than 1 items, move the last item to root
    // and call heapify.
    if (heap_size > 1)
    {
        harr[0] = harr[heap_size-1];
        MinHeapify(0);
    }
    heap_size--;

    return root;
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
```

```
    if (l < heap_size && harr[l] < harr[i])
        smallest = l;
    if (r < heap_size && harr[r] < harr[smallest])
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}

// A utility function to swap two elements
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// Function to return k'th smallest element in a given array
int kthSmallest(int arr[], int n, int k)
{
    // Build a heap of n elements: O(n) time
    MinHeap mh(arr, n);

    // Do extract min (k-1) times
    for (int i=0; i<k-1; i++)
        mh.extractMin();

    // Return root
    return mh.getMin();
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 19};
    int n = sizeof(arr)/sizeof(arr[0]), k = 2;
    cout << "K'th smallest element is " << kthSmallest(arr, n, k);
    return 0;
}
```

Output:

```
K'th smallest element is 5
```

Time complexity of this solution is O(n + kLogn).

Method 3 (Using Max-Heap)
We can also use Max Heap for finding the k'th smallest element. Following is algorithm.
1) Build a Max-Heap MH of the first k elements (arr[0] to arr[k-1]) of the given array. O(k)

2) For each element, after the k'th element (arr[k] to arr[n-1]), compare it with root of MH.
……a) If the element is less than the root then make it root and call heapify for MH
……b) Else ignore it.
// The step 2 is O((n-k)*logk)

3) Finally, root of the MH is the kth smallest element.

Time complexity of this solution is O(k + (n-k)*Logk)

The following is C++ implementation of above algorithm

```
 // A C++ program to find k'th smallest element using max heap
#include<iostream>
#include<climits>
using namespace std;

// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Max Heap
class MaxHeap
{
    int *harr; // pointer to array of elements in heap
    int capacity; // maximum possible size of max heap
    int heap_size; // Current number of elements in max heap
public:
    MaxHeap(int a[], int size); // Constructor
    void maxHeapify(int i);  //To maxHeapify subtree rooted with index i
    int parent(int i) { return (i-1)/2; }
    int left(int i) { return (2*i + 1); }
    int right(int i) { return (2*i + 2); }

    int extractMax();  // extracts root (maximum) element
    int getMax() { return harr[0]; } // Returns maximum

    // to replace root with new node x and heapify() new root
    void replaceMax(int x) { harr[0] = x;  maxHeapify(0); }
};

MaxHeap::MaxHeap(int a[], int size)
{
    heap_size = size;
    harr = a;  // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
```

```
        maxHeapify(i);
        i--;
    }
}

// Method to remove maximum element (or root) from max heap
int MaxHeap::extractMax()
{
    if (heap_size == 0)
        return INT_MAX;

    // Store the maximum vakue.
    int root = harr[0];

    // If there are more than 1 items, move the last item to root
    // and call heapify.
    if (heap_size > 1)
    {
        harr[0] = harr[heap_size-1];
        maxHeapify(0);
    }
    heap_size--;

    return root;
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MaxHeap::maxHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int largest = i;
    if (l < heap_size && harr[l] > harr[i])
        largest = l;
    if (r < heap_size && harr[r] > harr[largest])
        largest = r;
    if (largest != i)
    {
        swap(&harr[i], &harr[largest]);
        maxHeapify(largest);
    }
}

// A utility function to swap two elements
void swap(int *x, int *y)
{
    int temp = *x;
```

```
    *x = *y;
    *y = temp;
}

// Function to return k'th largest element in a given array
int kthSmallest(int arr[], int n, int k)
{
    // Build a heap of first k elements: O(k) time
    MaxHeap mh(arr, k);

    // Process remaining n-k elements.  If current element is
    // smaller than root, replace root with current element
    for (int i=k; i<n; i++)
        if (arr[i] < mh.getMax())
           mh.replaceMax(arr[i]);

    // Return root
    return mh.getMax();
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 19};
    int n = sizeof(arr)/sizeof(arr[0]), k = 4;
    cout << "K'th smallest element is " <<  kthSmallest(arr, n, k);
    return 0;
}
```

Output:

```
K'th smallest element is 5
```

Method 4 (QuickSelect)
This is an optimization over method 1 if QuickSortis used as a sorting algorithm in first step. In QuickSort, we pick a pivot element, then move the pivot element to its correct position and partition the array around it. The idea is, not to do complete quicksort, but stop at the point where pivot itself is k'th smallest element. Also, not to recur for both left and right sides of pivot, but recur for one of them according to the position of pivot. The worst case time complexity of this method is O(n^2), but it works in O(n) on average.

C++

```
 #include<iostream>
#include<climits>
using namespace std;
```

```
int partition(int arr[], int l, int r);

// This function returns k'th smallest element in arr[l..r] using
// QuickSort based method.  ASSUMPTION: ALL ELEMENTS IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
    // If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1)
    {
        // Partition the array around last element and get
        // position of pivot element in sorted array
        int pos = partition(arr, l, r);

        // If position is same as k
        if (pos-l == k-1)
            return arr[pos];
        if (pos-l > k-1)  // If position is more, recur for left subarray
            return kthSmallest(arr, l, pos-1, k);

        // Else recur for right subarray
        return kthSmallest(arr, pos+1, r, k-pos+l-1);
    }

    // If k is more than number of elements in array
    return INT_MAX;
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Standard partition process of QuickSort().  It considers the last
// element as pivot and moves all smaller element to left of it
// and greater elements to right
int partition(int arr[], int l, int r)
{
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++)
    {
        if (arr[j] <= x)
        {
            swap(&arr[i], &arr[j]);
            i++;
        }
```

```
    }
    swap(&arr[i], &arr[r]);
    return i;
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 4, 19, 26};
    int n = sizeof(arr)/sizeof(arr[0]), k = 3;
    cout << "K'th smallest element is " << kthSmallest(arr, 0, n-1, k);
    return 0;
}
```

Java

```
 // Java code for kth smallest element in an array
import java.util.Arrays;
import java.util.Collections;

class GFG
{
    // Standard partition process of QuickSort.
    // It considers the last element as pivot
    // and moves all smaller element to left of
    // it and greater elements to right
    public static int partition(Integer [] arr, int l,
                                            int r)
    {
        int x = arr[r], i = l;
        for (int j = l; j <= r - 1; j++)
        {
            if (arr[j] <= x)
            {
                //Swapping arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;

                i++;
            }
        }

        //Swapping arr[i] and arr[r]
        int temp = arr[i];
        arr[i] = arr[r];
        arr[r] = temp;
```

```
        return i;
    }

    // This function returns k'th smallest element
    // in arr[l..r] using QuickSort based method.
    // ASSUMPTION: ALL ELEMENTS IN ARR[] ARE DISTINCT
    public static int kthSmallest(Integer[] arr, int l,
                                        int r, int k)
    {
        // If k is smaller than number of elements
        // in array
        if (k > 0 && k <= r - l + 1)
        {
            // Partition the array around last
            // element and get position of pivot
            // element in sorted array
            int pos = partition(arr, l, r);

            // If position is same as k
            if (pos-l == k-1)
                return arr[pos];

            // If position is more, recur for
            // left subarray
            if (pos-l > k-1)
                return kthSmallest(arr, l, pos-1, k);

            // Else recur for right subarray
            return kthSmallest(arr, pos+1, r, k-pos+l-1);
        }

        // If k is more than number of elements
        // in array
        return Integer.MAX_VALUE;
    }

    // Driver program to test above methods
    public static void main(String[] args)
    {
        Integer arr[] = new Integer[]{12, 3, 5, 7, 4, 19, 26};
        int k = 3;
        System.out.print( "K'th smallest element is " +
                     kthSmallest(arr, 0, arr.length - 1, k) );
    }
}

// This code is contributed by Chhavi
```

Output:

```
K'th smallest element is 5
```

There are two more solutions which are better than above discussed ones: One solution is to do randomized version of quickSelect() and other solution is worst case linear time algorithm (see the following posts).

K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)
K'th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time)

References:
http://www.ics.uci.edu/~eppstein/161/960125.html
http://www.cs.rit.edu/~ib/Classes/CS515_Spring12-13/Slides/022-SelectMasterThm.pdf

Improved By : nitin mittal, vt_m

Source

https://www.geeksforgeeks.org/kth-smallestlargest-element-unsorted-array/

Chapter 132

K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)

K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time) - GeeksforGeeks

We recommend to read following post as a prerequisite of this post.

K'th Smallest/Largest Element in Unsorted Array | Set 1

Given an array and a number k where k is smaller than size of array, we need to find the k'th largest element in the given array. It is given that ll array elements are distinct.

Examples:

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 3
Output: 7

Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 4
Output: 10
```

We have discussed three different solutions here.

In this post method 4 is discussed which is mainly an extension of method 3 (QuickSelect) discussed in the previouspost.

```
 #include<iostream>
#include<climits>
#include<cstdlib>
```

```
using namespace std;

int randomPartition(int arr[], int l, int r);

// This function returns k'th smallest element in arr[l..r] using
// QuickSort based method.  ASSUMPTION: ALL ELEMENTS IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
    // If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1)
    {
        // Partition the array around last element and get
        // position of pivot element in sorted array
        int pos = randomPartition(arr, l, r);

        // If position is same as k
        if (pos-l == k-1)
            return arr[pos];
        if (pos-l > k-1)  // If position is more, recur for left subarray
            return kthSmallest(arr, l, pos-1, k);

        // Else recur for right subarray
        return kthSmallest(arr, pos+1, r, k-pos+l-1);
    }

    // If k is more than number of elements in array
    return INT_MAX;
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Standard partition process of QuickSort().  It considers the last
// element as pivot and moves all smaller element to left of it
// and greater elements to right
int partition(int arr[], int l, int r)
{
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++)
    {
        if (arr[j] <= x)
        {
            swap(&arr[i], &arr[j]);
            i++;
```

```
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}

int randomPartition(int arr[], int l, int r)
{
    int n = r-l+1;
    int pivot = rand() % n;
    swap(&arr[l + pivot], &arr[r]);
    return partition(arr, l, r);
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 4, 19, 26};
    int n = sizeof(arr)/sizeof(arr[0]), k = 3;
    cout << "K'th smallest element is " << kthSmallest(arr, 0, n-1, k);
    return 0;
}
```

References:
https://www.geeksforgeeks.org/kth-smallestlargest-element-unsorted-array/

Source

https://www.geeksforgeeks.org/kth-smallestlargest-element-unsorted-array-set-2-expected-linear-time-2/

Chapter 133

K'th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time)

K'th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time) - GeeksforGeeks

We recommend reading following posts as a prerequisite of this post.

K'th Smallest/Largest Element in Unsorted Array | Set 1
K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)

Given an array and a number k where k is smaller than the size of the array, we need to find the k'th smallest element in the given array. It is given that all array elements are distinct.

Examples:

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 3
Output: 7

Input: arr[] = {7, 10, 4, 3, 20, 15}
       k = 4
Output: 10
```

In previous post, we discussed an expected linear time algorithm. In this post, a worst-case linear time method is discussed. *The idea in this new method is similar to quickSelect(), we get worst-case linear time by selecting a pivot that divides array in a balanced way (there are not very few elements on one side and many on another side).* After the array is divided in a balanced way, we apply the same steps as used in quickSelect() to decide whether to go left or right of the pivot.

Following is complete algorithm.

```
kthSmallest(arr[0..n-1], k)
1) Divide arr[] into  n/5  groups where size of each group is 5
   except possibly the last group which may have less than 5 elements.

2) Sort the above created  n/5  groups and find median
   of all groups. Create an auxiliary array 'median[]' and store medians
   of all  n/5  groups in this median array.

// Recursively call this method to find median of median[0.. n/5 -1]
3) medOfMed = kthSmallest(median[0.. n/5 -1],  n/10 )

4) Partition arr[] around medOfMed and obtain its position.
     pos = partition(arr, n, medOfMed)

5) If pos == k return medOfMed
6) If pos > k return kthSmallest(arr[l..pos-1], k)
7) If pos < k return kthSmallest(arr[pos+1..r], k-pos+l-1)
```

In above algorithm, last 3 steps are same as algorithm in previous post. The first four steps are used to obtain a good point for partitioning the array (to make sure that there are not too many elements either side of pivot).

Following is C++ implementation of above algorithm.

```
 // C++ implementation of worst case linear time algorithm
// to find k'th smallest element
#include<iostream>
#include<algorithm>
#include<climits>

using namespace std;

int partition(int arr[], int l, int r, int k);

// A simple function to find median of arr[].  This is called
// only for an array of size 5 in this program.
int findMedian(int arr[], int n)
{
    sort(arr, arr+n);  // Sort the array
    return arr[n/2];   // Return middle element
}

// Returns k'th smallest element in arr[l..r] in worst case
// linear time. ASSUMPTION: ALL ELEMENTS IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
    // If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1)
```

```
    {
        int n = r-l+1; // Number of elements in arr[l..r]

        // Divide arr[] in groups of size 5, calculate median
        // of every group and store it in median[] array.
        int i, median[(n+4)/5]; // There will be floor((n+4)/5) groups;
        for (i=0; i<n/5; i++)
            median[i] = findMedian(arr+l+i*5, 5);
        if (i*5 < n) //For last group with less than 5 elements
        {
            median[i] = findMedian(arr+l+i*5, n%5);
            i++;
        }

        // Find median of all medians using recursive call.
        // If median[] has only one element, then no need
        // of recursive call
        int medOfMed = (i == 1)? median[i-1]:
                                 kthSmallest(median, 0, i-1, i/2);

        // Partition the array around a random element and
        // get position of pivot element in sorted array
        int pos = partition(arr, l, r, medOfMed);

        // If position is same as k
        if (pos-l == k-1)
            return arr[pos];
        if (pos-l > k-1)  // If position is more, recur for left
            return kthSmallest(arr, l, pos-1, k);

        // Else recur for right subarray
        return kthSmallest(arr, pos+1, r, k-pos+l-1);
    }

    // If k is more than number of elements in array
    return INT_MAX;
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// It searches for x in arr[l..r], and partitions the array
// around x.
int partition(int arr[], int l, int r, int x)
```

```
{
    // Search for x in arr[l..r] and move it to end
    int i;
    for (i=l; i<r; i++)
        if (arr[i] == x)
           break;
    swap(&arr[i], &arr[r]);

    // Standard partition algorithm
    i = l;
    for (int j = l; j <= r - 1; j++)
    {
        if (arr[j] <= x)
        {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}

// Driver program to test above methods
int main()
{
    int arr[] = {12, 3, 5, 7, 4, 19, 26};
    int n = sizeof(arr)/sizeof(arr[0]), k = 3;
    cout << "K'th smallest element is "
         << kthSmallest(arr, 0, n-1, k);
    return 0;
}
```

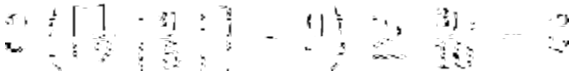
Output:

```
K'th smallest element is 5
```

Time Complexity:
The worst case time complexity of the above algorithm is O(n). Let us analyze all steps.

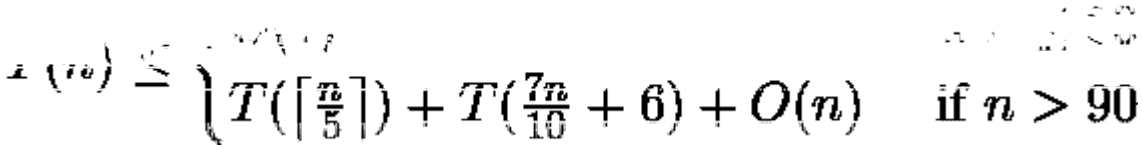
The steps 1) and 2) take O(n) time as finding median of an array of size 5 takes O(1) time and there are n/5 arrays of size 5.
The step 3) takes T(n/5) time. The step 4 is standard partition and takes O(n) time.
The interesting steps are 6) and 7). At most, one of them is executed. These are recursive steps. What is the worst case size of these recursive calls. The answer is maximum number of elements greater than medOfMed (obtained in step 3) or maximum number of elements smaller than medOfMed.
How many elements are greater than medOfMed and how many are smaller?
At least half of the medians found in step 2 are greater than or equal to medOfMed. Thus, at

least half of the n/5 groups contribute 3 elements that are greater than medOfMed, except for the one group that has fewer than 5 elements. Therefore, the number of elements greater than medOfMed is at least.



Similarly, the number of elements that are less than medOfMed is at least 3n/10 – 6. In the worst case, the function recurs for at most n – (3n/10 – 6) which is 7n/10 + 6 elements.

Note that 7n/10 + 6 < n for n > 20 20 and that any input of 80 or fewer elements requires O(1) time. We can therefore obtain the recurrence



$$T(n) \leq \begin{cases} T(\lceil \frac{n}{5} \rceil) + T(\frac{7n}{10} + 6) + O(n) & \text{if } n > 90 \end{cases}$$

We show that the running time is linear by substitution. Assume that T(n) cn for some constant c and all n > 80. Substituting this inductive hypothesis into the right-hand side of the recurrence yields

```
T(n)  <= cn/5 + c(7n/10 + 6) + O(n)
     <= cn/5 + c + 7cn/10 + 6c + O(n)
    <= 9cn/10 + 7c + O(n)
    <= cn,
```

since we can pick c large enough so that c(n/10 – 7) is larger than the function described by the O(n) term for all n > 80. The worst-case running time of is therefore linear (Source: http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap10.htm).

Note that the above algorithm is linear in worst case, but the constants are very high for this algorithm. Therefore, this algorithm doesn't work well in practical situations, randomized quickSelect works much better and preferred.

Sources:
MIT Video Lecture on Order Statistics, Median
Introduction to Algorithms by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.
http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap10.htm

This article is contributed by **Shivam**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : falcon95

Source

https://www.geeksforgeeks.org/kth-smallestlargest-element-unsorted-array-set-3-worst-case-linear-time/

Chapter 134

K'th largest element in a stream

K'th largest element in a stream - GeeksforGeeks

Given an infinite stream of integers, find the k'th largest element at any point of time.

Example:

```
Input:
stream[] = {10, 20, 11, 70, 50, 40, 100, 5, ...}
k = 3

Output:    {_,   _, 10, 11, 20, 40, 50,  50, ...}
```

Extra space allowed is O(k).

We have discussed different approaches to find k'th largest element in an array in the following posts.
K'th Smallest/Largest Element in Unsorted Array | Set 1
K'th Smallest/Largest Element in Unsorted Array | Set 2 (Expected Linear Time)
K'th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time)

Here we have a stream instead of whole array and we are allowed to store only k elements.

A **Simple Solution** is to keep an array of size k. The idea is to keep the array sorted so that the k'th largest element can be found in O(1) time (we just need to return first element of array if array is sorted in increasing order)
How to process a new element of stream?
For every new element in stream, check if the new element is smaller than current k'th largest element. If yes, then ignore it. If no, then remove the smallest element from array and insert new element in sorted order. Time complexity of processing a new element is O(k).

A **Better Solution** is to use a Self Balancing Binary Search Tree of size k. The k'th largest element can be found in O(Logk) time.

How to process a new element of stream?
For every new element in stream, check if the new element is smaller than current k'th largest element. If yes, then ignore it. If no, then remove the smallest element from the tree and insert new element. Time complexity of processing a new element is O(Logk).

An **Efficient Solution** is to use Min Heap of size k to store k largest elements of stream. The k'th largest element is always at root and can be found in O(1) time.
How to process a new element of stream?
Compare the new element with root of heap. If new element is smaller, then ignore it. Otherwise replace root with new element and call heapify for the root of modified heap. Time complexity of finding the k'th largest element is O(Logk).

```
 // A C++ program to find k'th smallest element in a stream
#include<iostream>
#include<climits>
using namespace std;

// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Min Heap
class MinHeap
{
    int *harr; // pointer to array of elements in heap
    int capacity; // maximum possible size of min heap
    int heap_size; // Current number of elements in min heap
public:
    MinHeap(int a[], int size); // Constructor
    void buildHeap();
    void MinHeapify(int i);  //To minheapify subtree rooted with index i
    int parent(int i)  { return (i-1)/2;  }
    int left(int i)    { return (2*i + 1);  }
    int right(int i)   { return (2*i + 2);  }
    int extractMin();  // extracts root (minimum) element
    int getMin()       {  return harr[0]; }

    // to replace root with new node x and heapify() new root
    void replaceMin(int x) { harr[0] = x; MinHeapify(0); }
};

MinHeap::MinHeap(int a[], int size)
{
    heap_size = size;
    harr = a;  // store address of array
}

void MinHeap::buildHeap()
{
    int i = (heap_size - 1)/2;
```

```
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// Method to remove minimum element (or root) from min heap
int MinHeap::extractMin()
{
    if (heap_size == 0)
        return INT_MAX;

    // Store the minimum vakue.
    int root = harr[0];

    // If there are more than 1 items, move the last item to root
    // and call heapify.
    if (heap_size > 1)
    {
        harr[0] = harr[heap_size-1];
        MinHeapify(0);
    }
    heap_size--;

    return root;
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && harr[l] < harr[i])
        smallest = l;
    if (r < heap_size && harr[r] < harr[smallest])
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}

// A utility function to swap two elements
void swap(int *x, int *y)
```

```
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// Function to return k'th largest element from input stream
void kthLargest(int k)
{
    // count is total no. of elements in stream seen so far
    int count = 0, x;  // x is for new element

    // Create a min heap of size k
    int *arr = new int[k];
    MinHeap mh(arr, k);


    while (1)
    {
        // Take next element from stream
        cout << "Enter next element of stream ";
        cin >> x;

        // Nothing much to do for first k-1 elements
        if (count < k-1)
        {
            arr[count] = x;
            count++;
        }

        else
        {
          // If this is k'th element, then store it
          // and build the heap created above
          if (count == k-1)
          {
              arr[count] = x;
              mh.buildHeap();
          }

          else
          {
               // If next element is greater than
               // k'th largest, then replace the root
               if (x > mh.getMin())
                  mh.replaceMin(x); // replaceMin calls
                                    // heapify()
          }
```

```
            // Root of heap is k'th largest element
            cout << "K'th largest element is "
                 << mh.getMin() << endl;
            count++;
        }
    }
}

// Driver program to test above methods
int main()
{
    int k = 3;
    cout << "K is " << k << endl;
    kthLargest(k);
    return 0;
}
```

Output

```
K is 3
Enter next element of stream 23
Enter next element of stream 10
Enter next element of stream 15
K'th largest element is 10
Enter next element of stream 70
K'th largest element is 15
Enter next element of stream 5
K'th largest element is 15
Enter next element of stream 80
K'th largest element is 23
Enter next element of stream 100
K'th largest element is 70
Enter next element of stream
CTRL + C pressed
```

This article is contributed by **Shivam Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

https://www.geeksforgeeks.org/kth-largest-element-in-a-stream/

Chapter 135

Largest gap in an array

Largest gap in an array - GeeksforGeeks

Given an unsorted array of length N and we have to find largest gap between any two elements of array.In simple words, find max($|A_i - A_j|$) where 1 i N and 1 j N.

Examples:

```
Input : arr = {3, 10, 6, 7}
Output : 7
Explanation :
Here, we can see largest gap can be
found between 3 and 10 which is 7

Input : arr = {-3, -1, 6, 7, 0}
Output : 10
Explanation :
Here, we can see largest gap can be
found between -3 and 7 which is 10
```

Simple Approach:
A simple solution is, we can use naive approach.We will check absolute difference of every pair in the array and we will find the maximum value of it.So we will run two loops one is for i and one is for j complexity of this method is O(N^2)

C

```
 // A C program to find largest gap
// between two elements in an array.
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
```

```
// function to solve the given problem
int solve(int a[], int n)
{
    int max1 = INT_MIN;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (abs(a[i] - a[j]) > max1) {
                max1 = abs(a[i] - a[j]);
            }
        }
    }
    return max1;
}

int main()
{
    int arr[] = { -1, 2, 3, -4, -10, 22 };
    int size = sizeof(arr) / sizeof(arr[0]);
    printf("Largest gap is : %d", solve(arr, size));
    return 0;
}
```

Java

```
 // A Java program to find
// largest gap between
// two elements in an array.
import java .io.*;

class GFG
{

// function to solve
// the given problem
static int solve(int []a,
                 int n)
{
    int max1 = Integer.MIN_VALUE ;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (Math.abs(a[i] -
                         a[j]) > max1)
            {
                max1 = Math.abs(a[i] -
                                a[j]);
            }
```

```
        }
    }
    return max1;
}

// Driver Code
static public void main (String[] args)
{
    int []arr = {-1, 2, 3,
                 -4, -10, 22};
    int size = arr.length;
    System.out.println("Largest gap is : " +
                        solve(arr, size));
}
}

// This code is contributed
// by anuj_67.
```

C#

```
 // A C# program to find
// largest gap between
// two elements in an array.
using System;

class GFG
{

// function to solve
// the given problem
static int solve(int []a,
                 int n)
{
    int max1 = int.MinValue ;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (Math.Abs(a[i] -
                         a[j]) > max1)
            {
                max1 = Math.Abs(a[i] -
                                a[j]);
            }
        }
    }
    return max1;
```

```
}

// Driver Code
static public void Main ()
{
    int []arr = {-1, 2, 3,
                 -4, -10, 22};
    int size = arr.Length;
    Console.WriteLine("Largest gap is : " +
                          solve(arr, size));
}
}

// This code is contributed
// by anuj_67.
```

PHP

```
 <?php
// A PHP program to find
// largest gap between
// two elements in an array.

// function to solve
// the given problem
function solve($a, $n)
{
    $max1 = PHP_INT_MIN;
    for ($i = 0; $i < $n; $i++)
    {
        for ($j = 0; $j < $n; $j++)
        {
            if (abs($a[$i] -
                    $a[$j]) > $max1)
            {
                $max1 = abs($a[$i] -
                            $a[$j]);
            }
        }
    }
    return $max1;
}

// Driver Code
$arr = array(-1, 2, 3,
             -4, -10, 22);
$size = count($arr);
echo "Largest gap is : ",
```

```
      solve($arr, $size);

// This code is contributed
// by anuj_67.
?>
```

Output:

```
Largest gap is : 32
```

Better Approach:
Now we will see a better approach it is greedy approach which can solve this problem in O(N). we will find maximum and minimum element of the array which can be done in O(N) and then we will return value of (maximum-minimum).

C

```
 // A C program to find largest gap between
// two elements in an array.
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>

// function to solve the given problem
int solve(int a[], int n)
{
    int min1 = a[0];
    int max1 = a[0];

    // finding maximum and minimum of an array
    for (int i = 0; i < n; i++) {
        if (a[i] > max1)
            max1 = a[i];
        if (a[i] < min1)
            min1 = a[i];
    }

    return abs(min1 - max1);
}

int main()
{
    int arr[] = { -1, 2, 3, 4, -10 };
    int size = sizeof(arr) / sizeof(arr[0]);
    printf("Largest gap is : %d", solve(arr, size));
    return 0;
}
```

Java

```
 // A Java program to find largest gap
// between two elements in an array.
import java.io.*;

class GFG {

    // function to solve the given
    // problem
    static int solve(int a[], int n)
    {
        int min1 = a[0];
        int max1 = a[0];

        // finding maximum and minimum
        // of an array
        for (int i = 0; i < n; i++)
        {
            if (a[i] > max1)
                max1 = a[i];
            if (a[i] < min1)
                min1 = a[i];
        }

        return Math.abs(min1 - max1);
    }

    // Driver code
    public static void main (String[] args)
    {
        int []arr = { -1, 2, 3, 4, -10 };
        int size = arr.length;
        System.out.println("Largest gap is : "
                        + solve(arr, size));
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // A C# program to find
// largest gap between
// two elements in an array.
using System;
```

```
class GFG
{

    // function to solve
    // the given problem
    static int solve(int []a,
                     int n)
    {
        int min1 = a[0];
        int max1 = a[0];

        // finding maximum and
        // minimum of an array
        for (int i = 0; i < n; i++)
        {
            if (a[i] > max1)
                max1 = a[i];
            if (a[i] < min1)
                min1 = a[i];
        }

        return Math.Abs(min1 -
                        max1);
    }

    // Driver code
    public static void Main ()
    {
        int []arr = {-1, 2, 3, 4, -10};
        int size = arr.Length;
        Console.WriteLine("Largest gap is : " +
                              solve(arr, size));
    }
}

// This code is contributed
// by anuj_67.
```

PHP

```
 <?php
// A PHP program to find
// largest gap between
// two elements in an array.

// function to solve
// the given problem
function solve($a, $n)
```

```
{
    $min1 = $a[0];
    $max1 = $a[0];

    // finding maximum and
    // minimum of an array
    for ($i = 0; $i < $n; $i++)
    {
        if ($a[$i] > $max1)
            $max1 = $a[$i];
        if ($a[$i] < $min1)
            $min1 = $a[$i];
    }

    return abs($min1 - $max1);
}

// Driver Code
$arr = array(-1, 2, 3, 4, -10);
$size = count($arr);
echo "Largest gap is : ",
      solve($arr, $size);

// This code is contributed
// by anuj_67.
?>
```

Output:

```
Largest gap is : 14
```

Improved By : vt_m

Source

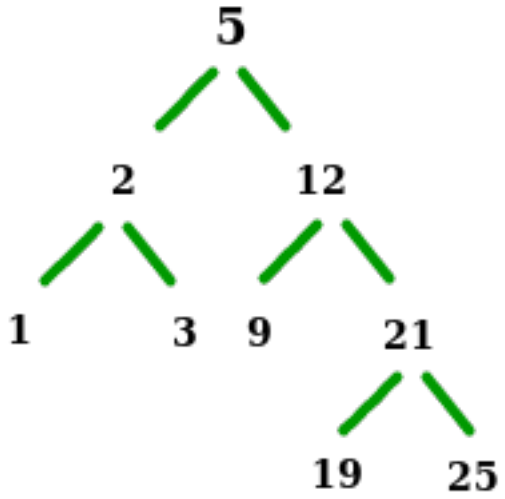
https://www.geeksforgeeks.org/largest-gap-in-an-array/

Chapter 136

Largest number less than or equal to N in BST (Iterative Approach)

Largest number less than or equal to N in BST (Iterative Approach) - GeeksforGeeks

We have a binary search tree and a number N. Our task is to find the greatest number in the binary search tree that is less than or equal to N. Print the value of the element if it exists otherwise print -1.



Examples:For the above given binary search tree-

```
Input : N = 24
Output :result = 21
```

```
(searching for 24 will be like-5->12->21)

Input  : N = 4
Output : result = 3
(searching for 4 will be like-5->2->3)
```

We have discussed recursive approach in below post.
Largest number in BST which is less than or equal to N

Here an iterative approach is discussed. We try to find the predecessor of the target. Keep two pointers, one pointing to the current node and one for storing the answer. If the current node's data > N, we move towards left. In other case, when current node's data is less than N, the current node can be our answer (so far), and we move towards right.

```
 // C++ code to find the largest value smaller
// than or equal to N
#include <bits/stdc++.h>
using namespace std;

struct Node {
    int key;
    Node *left, *right;
};

// To create new BST Node
Node* newNode(int item)
{
    Node* temp = new Node;
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

// To insert a new node in BST
Node* insert(Node* node, int key)
{
    // if tree is empty return new node
    if (node == NULL)
        return newNode(key);

    // if key is less then or grater then
    // node value then recur down the tree
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    // return the (unchanged) node pointer
```

```
    return node;
}

// Returns largest value smaller than or equal to
// key. If key is smaller than the smallest, it
// returns -1.
int findFloor(Node* root, int key)
{
    Node *curr = root, *ans = NULL;
    while (curr) {
        if (curr->key <= key) {
            ans = curr;
            curr = curr->right;
        }
        else
            curr = curr->left;
    }
    if (ans)
        return ans->key;
    return -1;
}

// Driver code
int main()
{
    int N = 25;

    Node* root = insert(root, 19);
    insert(root, 2);
    insert(root, 1);
    insert(root, 3);
    insert(root, 12);
    insert(root, 9);
    insert(root, 21);
    insert(root, 19);
    insert(root, 25);

    printf("%d", findFloor(root, N));

    return 0;
}
```

Output:

```
25
```

Source

https://www.geeksforgeeks.org/largest-number-less-equal-n-bst-iterative-approach/

Chapter 137

Largest subarray having sum greater than k

Largest subarray having sum greater than k - GeeksforGeeks

Given an array of integers and a value k, find length of largest subarray having sum greater than k.

Examples:

```
Input : arr[] = {-2, 1, 6, -3}, k = 5
Output : 2
Largest subarray with sum greater than
5 is  {1, 6}.

Input : arr[] = {2, -3, 3, 2, 0, -1}, k = 3
Output : 5
Largest subarray with sum greater than
3 is {2, -3, 3, 2, 0}.
```

A **simple** solution is to one by one consider each subarray and find its sum. If sum is greater than k, then compare length of this subarray with maximum length found so far. Time complexity of this solution is $O(n^2)$.

An **efficient** solution is to use prefix sum and binary search. The idea is to traverse the array and store prefix sum and corresponding array index in a vector of pairs. After finding prefix sum, sort the vector in increasing order of prefix sum, and for same value of prefix sum, sort according to index. Create another array minInd[], in which minInd[i] stores the minimum index value in range [0..i] in sorted prefix sum vector. After this, start traversing array arr[] and store sum of subarray arr[0..i] in sum. If sum is greater than k, then largest subarray having sum greater than k is arr[0..i] having length i + 1. If sum is less than or equal to k, then a value greater than or equal to k + 1 – sum has to be added to sum to make

it at least k+1. Let this value be x. To add x to sum, -x can be subtracted from it because sum-(-x) = sum + x. So a prefix array arr[0..j] (j<i), is needed to be found having sum at most -x (at most -x because k+1-sum is least value, now its negative is taken so it will be maximum value allowed). The resultant subarray arr[j+1..i] should be as large as possible. For this the value of j should be as minimum as possible. Thus the problem reduces to finding a prefix sum having value at most -x and its ending index should be minimum. To find the prefix sum, binary search can be performed on prefix sum vector. Let the index ind denotes that in prefix sum vector all prefix sum values upto index ind are less than or equal to -x. The minimum index value in range[0..ind] is minInd[ind]. If minInd[ind] is greater than i, then no subarray exists having sum -x in range[0..i-1]. Else arr[minInd[ind]+1..i] has sum greater than k. Compare its length with maximum length found so far.

Below is the implementation of above approach:

```
 // CPP program to find largest subarray
// having sum greater than k.
#include <bits/stdc++.h>
using namespace std;

// Comparison function used to sort preSum vector.
bool compare(const pair<int, int>& a,
             const pair<int, int>& b)
{
    if (a.first == b.first)
        return a.second < b.second;

    return a.first < b.first;
}

// Function to find index in preSum vector upto which
// all prefix sum values are less than or equal to val.
int findInd(vector<pair<int, int> >& preSum, int n,
                                            int val)
{

    // Starting and ending index of search space.
    int l = 0;
    int h = n - 1;
    int mid;

    // To store required index value.
    int ans = -1;

    // If middle value is less than or equal to
    // val then index can lie in mid+1..n
    // else it lies in 0..mid-1.
    while (l <= h) {
        mid = (l + h) / 2;
        if (preSum[mid].first <= val) {
```

```
            ans = mid;
            l = mid + 1;
        }
        else
            h = mid - 1;
    }

    return ans;
}

// Function to find largest subarray having sum
// greater than or equal to k.
int largestSub(int arr[], int n, int k)
{
    int i;

    // Length of largest subarray.
    int maxlen = 0;

    // Vector to store pair of prefix sum
    // and corresponding ending index value.
    vector<pair<int, int> > preSum;

    // To store current value of prefix sum.
    int sum = 0;

    // To store minimum index value in range
    // 0..i of preSum vector.
    int minInd[n];

    // Insert values in preSum vector.
    for (i = 0; i < n; i++) {
        sum = sum + arr[i];
        preSum.push_back({ sum, i });
    }

    sort(preSum.begin(), preSum.end(), compare);

    // Update minInd array.
    minInd[0] = preSum[0].second;

    for (i = 1; i < n; i++) {
        minInd[i] = min(minInd[i - 1], preSum[i].second);
    }

    sum = 0;
    for (i = 0; i < n; i++) {
        sum = sum + arr[i];
```

```
        // If sum is greater than k, then answer
        // is i+1.
        if (sum > k)
            maxlen = i + 1;

        // If sum is less than or equal to k, then
        // find if there is a prefix array having sum
        // that needs to be added to current sum to
        // make its value greater than k. If yes, then
        // compare length of updated subarray with
        // maximum length found so far.
        else {
            int ind = findInd(preSum, n, sum - k - 1);
            if (ind != -1 && minInd[ind] < i)
                maxlen = max(maxlen, i - minInd[ind]);
        }
    }

    return maxlen;
}

// Driver code.
int main()
{
    int arr[] = { -2, 1, 6, -3 };
    int n = sizeof(arr) / sizeof(arr[0]);

    int k = 5;

    cout << largestSub(arr, n, k);
    return 0;
}
```

Output:

```
2
```

Time Complexity: O(nlogn)
Auxiliary Space: O(n)

Source

https://www.geeksforgeeks.org/largest-subarray-having-sum-greater-than-k/

Chapter 138

Last duplicate element in a sorted array

Last duplicate element in a sorted array - GeeksforGeeks

We have a sorted array with duplicate elements and we have to find the index of last duplicate element and print index of it and also print the duplicate element. If no such element found print a message.
Examples:

```
Input : arr[] = {1, 5, 5, 6, 6, 7}
Output :
Last index: 4
Last duplicate item: 6

Input : arr[] = {1, 2, 3, 4, 5}
Output : No duplicate found
```

We simply iterate through the array in reverse order and compare the current and previous element. If a match is found then we print the index and duplicate element. As this is sorted array it will be the last duplicate. If no such element is found we will print the message for it.

```
1- for i = n-1 to 0
     if (arr[i] == arr[i-1])
        Print current element and its index.
        Return
2- If no such element found print a message
   of no duplicate found.
```

C++

```
 // To print last duplicate element and its
// index in a sorted array
#include <stdio.h>

void dupLastIndex(int arr[], int n) {

  // if array is null or size is less
  // than equal to 0 return
  if (arr == NULL || n <= 0)
    return;

  // compare elements and return last
  // duplicate and its index
  for (int i = n - 1; i > 0; i--) {
    if (arr[i] == arr[i - 1]) {
      printf("Last index: %d\nLast "
            "duplicate item: %d\n", i, arr[i]);
      return;
    }
  }

  // If we reach here, then no duplicate
  // found.
  printf("no duplicate found");
}

int main() {
  int arr[] = {1, 5, 5, 6, 6, 7, 9};
  int n = sizeof(arr) / sizeof(int);
  dupLastIndex(arr, n);
  return 0;
}
```

Java

```
 // Java code to print last duplicate element
// and its index in a sorted array
import java.io.*;

class GFG
{
    static void dupLastIndex(int arr[], int n)
    {
        // if array is null or size is less
        // than equal to 0 return
```

```
        if (arr == null || n <= 0)
            return;

        // compare elements and return last
        // duplicate and its index
        for (int i = n - 1; i > 0; i--)
        {
            if (arr[i] == arr[i - 1])
            {
            System.out.println("Last index:" + i);
            System.out.println("Last duplicate item: "
                               + arr[i]);
            return;
            }
        }

        // If we reach here, then no duplicate
        // found.
        System.out.print("no duplicate found");
    }

    // Driver code
    public static void main (String[] args)
    {
        int arr[] = {1, 5, 5, 6, 6, 7, 9};
        int n = arr.length;
        dupLastIndex(arr, n);

    }
}

// This code is contributed by vt_m
```

Python3

```
 # Python3 code to print last duplicate
# element and its index in a sorted array

def dupLastIndex(arr, n):

    # if array is null or size is less
    # than equal to 0 return
    if (arr == None or n <= 0):
        return

    # compare elements and return last
    # duplicate and its index
    for i in range(n - 1, 0, -1):
```

```
        if (arr[i] == arr[i - 1]):
            print("Last index:", i, "\nLast",
                     "duplicate item:",arr[i])
            return

    # If we reach here, then no duplicate
    # found.
    print("no duplicate found")


arr = [1, 5, 5, 6, 6, 7, 9]
n = len(arr)
dupLastIndex(arr, n)

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# code to print last duplicate element
// and its index in a sorted array
using System;

class GFG {

    static void dupLastIndex(int []arr, int n)
    {

        // if array is null or size is less
        // than equal to 0 return
        if (arr == null || n <= 0)
            return;

        // compare elements and return last
        // duplicate and its index
        for (int i = n - 1; i > 0; i--)
        {
            if (arr[i] == arr[i - 1])
            {
                Console.WriteLine("Last index:" + i);
                Console.WriteLine("Last duplicate item: "
                                + arr[i]);
                return;
            }
        }

        // If we reach here, then no duplicate
```

```
        // found.
        Console.WriteLine("no duplicate found");
    }

    // Driver code
    public static void Main ()
    {
        int []arr = {1, 5, 5, 6, 6, 7, 9};
        int n = arr.Length;

        dupLastIndex(arr, n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to print last
// duplicate element and its
// index in a sorted array

function dupLastIndex($arr, $n)
{

    // if array is null or size is less
    // than equal to 0 return
    if ($arr == null or $n <= 0)
        return;

    // compare elements and return last
    // duplicate and its index
    for ( $i = $n - 1; $i > 0; $i--)
    {
        if ($arr[$i] == $arr[$i - 1])
        {
            echo "Last index:", $i , "\n";
            echo "Last duplicate item:", $arr[$i];
            return;
    }
}

    // If we reach here, then
    // no duplicate found.
    echo "no duplicate found";
}
```

```
// Driver Code
$arr = array(1, 5, 5, 6, 6, 7, 9);
$n = count($arr);
dupLastIndex($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
Last index: 4
Last duplicate item: 6
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/last-duplicate-element-sorted-array/

Chapter 139

Least frequent element in an array

Least frequent element in an array - GeeksforGeeks

Given an array, find the least frequent element in it. If there are multiple elements that appear least number of times, print any one of them.

Examples :

```
Input : arr[] = {1, 3, 2, 1, 2, 2, 3, 1}
Output : 3
3 appears minimum number of times in given
array.

Input : arr[] = {10, 20, 30}
Output : 10 or 20 or 30
```

A **simple solution** is to run two loops. The outer loop picks all elements one by one. The inner loop finds frequency of the picked element and compares with the minimum so far. Time complexity of this solution is $O(n^2)$

A **better solution** is to do sorting. We first sort the array, then linearly traverse the array.

C++

```
 // CPP program to find the least frequent element
// in an array.
#include <bits/stdc++.h>
using namespace std;

int leastFrequent(int arr[], int n)
{
```

```
    // Sort the array
    sort(arr, arr + n);

    // find the min frequency using linear traversal
    int min_count = n+1, res = -1, curr_count = 1;
    for (int i = 1; i < n; i++) {
        if (arr[i] == arr[i - 1])
            curr_count++;
        else {
            if (curr_count < min_count) {
                min_count = curr_count;
                res = arr[i - 1];
            }
            curr_count = 1;
        }
    }

    // If last element is least frequent
    if (curr_count < min_count)
    {
        min_count = curr_count;
        res = arr[n - 1];
    }

    return res;
}

// driver program
int main()
{
    int arr[] = {1, 3, 2, 1, 2, 2, 3, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << leastFrequent(arr, n);
    return 0;
}
```

Java

```
 // Java program to find the least frequent element
// in an array.
import java.io.*;
import java.util.*;

class GFG {

    static int leastFrequent(int arr[], int n)
    {
```

```
        // Sort the array
        Arrays.sort(arr);

        // find the min frequency using
        // linear traversal
        int min_count = n+1, res = -1;
        int curr_count = 1;

        for (int i = 1; i < n; i++) {
            if (arr[i] == arr[i - 1])
                curr_count++;
            else {
                if (curr_count < min_count) {
                    min_count = curr_count;
                    res = arr[i - 1];
                }

                curr_count = 1;
            }
        }

        // If last element is least frequent
        if (curr_count < min_count)
        {
            min_count = curr_count;
            res = arr[n - 1];
        }

        return res;
    }

    // driver program
    public static void main(String args[])
    {
        int arr[] = {1, 3, 2, 1, 2, 2, 3, 1};
        int n = arr.length;
        System.out.print(leastFrequent(arr, n));

    }
}

/*This code is contributed by Nikita Tiwari.*/
```

Python3

```
 # Python 3 program to find the least
# frequent element in an array.
```

```
def leastFrequent(arr, n) :

    # Sort the array
    arr.sort()

    # find the min frequency using
    # linear traversal
    min_count = n + 1
    res = -1
    curr_count = 1
    for i in range(1, n) :
        if (arr[i] == arr[i - 1]) :
            curr_count = curr_count + 1
        else :
            if (curr_count < min_count) :
                min_count = curr_count
                res = arr[i - 1]

            curr_count = 1


    # If last element is least frequent
    if (curr_count < min_count) :
        min_count = curr_count
        res = arr[n - 1]

    return res


# Driver program
arr = [1, 3, 2, 1, 2, 2, 3, 1]
n = len(arr)
print(leastFrequent(arr, n))


# This code is contributed
# by Nikita Tiwari.
```

C#

```
 // C# program to find the least
// frequent element in an array.
using System;

class GFG {

    static int leastFrequent(int[] arr, int n)
```

```
    {
        // Sort the array
        Array.Sort(arr);

        // find the min frequency
        // using linear traversal
        int min_count = n + 1, res = -1;
        int curr_count = 1;

        for (int i = 1; i < n; i++)
        {
            if (arr[i] == arr[i - 1])
                curr_count++;
            else
            {
                if (curr_count < min_count)
                {
                    min_count = curr_count;
                    res = arr[i - 1];
                }

                curr_count = 1;
            }
        }

        // If last element is least frequent
        if (curr_count < min_count)
        {
            min_count = curr_count;
            res = arr[n - 1];
        }

        return res;
    }

    // Driver code
    static public void Main ()
    {
        int[] arr = {1, 3, 2, 1, 2, 2, 3, 1};
        int n = arr.Length;

        // Function calling
        Console.Write(leastFrequent(arr, n));
    }
}

// This code is contributed by Shrikant13
```

PHP

```
 <?php
// PHP program to find the
// least frequent element
// in an array.

function leastFrequent($arr, $n)
{

    // Sort the array
    sort($arr);
    sort($arr , $n);

    // find the min frequency
    // using linear traversal
    $min_count = $n + 1;
    $res = -1;
    $curr_count = 1;
    for($i = 1; $i < $n; $i++)
    {
        if ($arr[$i] == $arr[$i - 1])
            $curr_count++;
        else
        {
            if ($curr_count < $min_count)
            {
                $min_count = $curr_count;
                $res = $arr[$i - 1];
            }
            $curr_count = 1;
        }
    }

    // If last element is
    // least frequent
    if ($curr_count < $min_count)
    {
        $min_count = $curr_count;
        $res = $arr[$n - 1];
    }

    return $res;
}

// Driver Code
{
    $arr = array(1, 3, 2, 1, 2, 2, 3, 1);
```

```
    $n = sizeof($arr) / sizeof($arr[0]);
    echo leastFrequent($arr, $n);
    return 0;
}

// This code is contributed by nitin mittal
?>
```

Output:

```
3
```

Time Complexity : O(n Log n)
Auxiliary Space : O(1)

An **efficient solution** is to use hashing. We create a hash table and store elements and their frequency counts as key value pairs. Finally we traverse the hash table and print the key with minimum value.

C++

```
 // CPP program to find the least frequent element
// in an array.
#include <bits/stdc++.h>
using namespace std;

int leastFrequent(int arr[], int n)
{
    // Insert all elements in hash.
    unordered_map<int, int> hash;
    for (int i = 0; i < n; i++)
        hash[arr[i]]++;

    // find the min frequency
    int min_count = n+1, res = -1;
    for (auto i : hash) {
        if (min_count >= i.second) {
            res = i.first;
            min_count = i.second;
        }
    }

    return res;
}

// driver program
int main()
```

```
{
    int arr[] = {1, 3, 2, 1, 2, 2, 3, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << leastFrequent(arr, n);
    return 0;
}
```

Java

```
 //Java program to find the least frequent element
//in an array
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

class GFG {

    static int leastFrequent(int arr[],int n)
    {

        // Insert all elements in hash.
        Map<Integer,Integer> count =
                   new HashMap<Integer,Integer>();

        for(int i = 0; i < n; i++)
        {
            int key = arr[i];
            if(count.containsKey(key))
            {
                int freq = count.get(key);
                freq++;
                count.put(key,freq);
            }
            else
                count.put(key,1);
        }

        // find min frequency.
        int min_count = n+1, res = -1;
        for(Entry<Integer,Integer> val : count.entrySet())
        {
            if (min_count >= val.getValue())
            {
                res = val.getKey();
                min_count = val.getValue();
            }
        }
```

```
        return res;
    }

    // driver program
    public static void main (String[] args) {

        int arr[] = {1, 3, 2, 1, 2, 2, 3, 1};
        int n = arr.length;

        System.out.println(leastFrequent(arr,n));
    }
}

// This code is contributed by Akash Singh.
```

C#

```
 // C# program to find the
// least frequent element
// in an array.
using System;
using System.Collections.Generic;

class GFG
{
    static int leastFrequent(int []arr,
                             int n)
    {
        // Insert all elements in hash.
        Dictionary<int, int> count =
                        new Dictionary<int,
                                       int>();
        for (int i = 0; i < n; i++)
        {
            int key = arr[i];
            if(count.ContainsKey(key))
            {
                int freq = count[key];
                freq++;
                count[key] = freq;
            }
            else
                count.Add(key, 1);
        }

        // find the min frequency
        int min_count = n + 1, res = -1;
        foreach (KeyValuePair<int,
```

```
                    int> pair in count)
        {
            if (min_count >= pair.Value)
            {
                res = pair.Key;
                min_count = pair.Value;
            }
        }
        return res;
    }

    // Driver Code
    static void Main()
    {
        int []arr = new int[]{1, 3, 2, 1,
                              2, 2, 3, 1};
        int n = arr.Length;
        Console.Write(least Frequent(arr, n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

```
3
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Improved By : shrikanth13, nitin mittal, manishshaw1

Source

https://www.geeksforgeeks.org/least-frequent-element-array/

Chapter 140

Linear Search

Linear Search - GeeksforGeeks

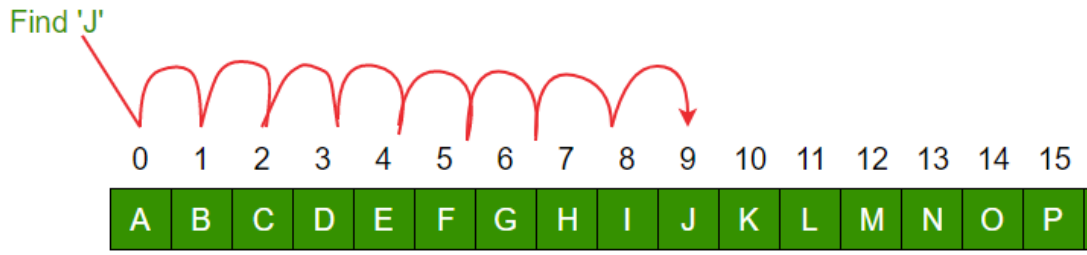
Problem: Given an array arr[] of n elements, write a function to search a given element x in arr[].

Examples :

```
Input : arr[] = {10, 20, 80, 30, 60, 50,
                 110, 100, 130, 170}
          x = 110;
Output : 6
Element x is present at index 6

Input : arr[] = {10, 20, 80, 30, 60, 50,
                 110, 100, 130, 170}
           x = 175;
Output : -1
Element x is not present in arr[].
```

A simple approach is to do **linear search**, i.e

- Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
- If x matches with an element, return the index.
- If x doesn't match with any of elements, return -1.



Example:

C/C++

```
 // C++ code for linearly search x in arr[].  If x
// is present  then return its  location,  otherwise
// return -1
int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```

Java

```
 // Java code for linearly search x in arr[].  If x
// is present  then return its  location,  otherwise
// return -1
class LinearSearch
{
    // This function returns index of element x in arr[]
    static int search(int arr[], int n, int x)
    {
        for (int i = 0; i < n; i++)
        {
            // Return the index of the element if the element
            // is found
            if (arr[i] == x)
                return i;
        }

        // return -1 if the element is not found
        return -1;
```

```
    }
}
```

Python

```
 # Python code for linearly search x in arr[].  If x
# is present then return its  location,  otherwise
# return -1
def search(arr, x):

    for i in range(len(arr)):

        if arr[i] == x:
            return i

    return -1
```

PHP

```
 <?php
// PHP code for linearly search
// x in arr[]. If x is present
// then return its location,
// otherwise return -1
function search($arr, $n, $x)
{
    $i;
    for ($i = 0; $i < $n; $i++)
        if ($arr[$i] == $x)
        return $i;
    return -1;
}

// This code is contributed
// by jit_t
?>
```

The time complexity of above algorithm is O(n).

Linear search is rarely used practically because other search algorithms such as the binary search algorithm and hash tables allow significantly faster searching comparison to Linear search.

Also See – Binary Search

Improved By : jit_t

Source

https://www.geeksforgeeks.org/linear-search/

Chapter 141

Linear Search vs Binary Search

Linear Search vs Binary Search - GeeksforGeeks

Prerequisite:

- Linear Search
- Binary Search

A linear search scans one item at a time, without jumping to any item .

1. The worst case complexity is O(n), sometimes known an O(n) search
2. Time taken to search elements keep increasing as the number of elements are increased.

A binary search however, cut down your search to half as soon as you find middle of a sorted list.

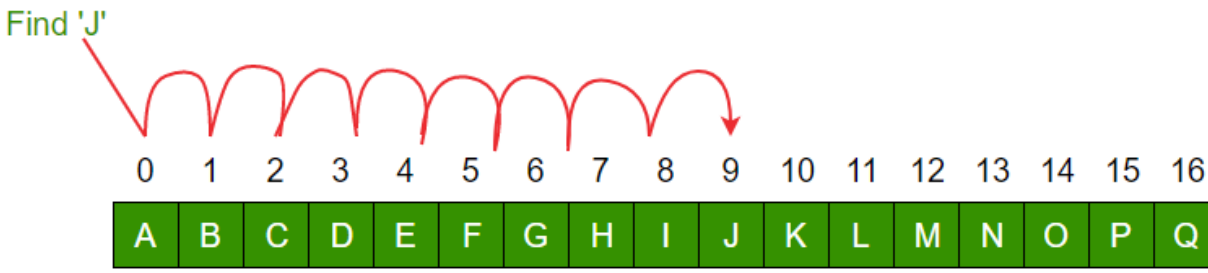
1. The middle element is looked to check if it is greater than or less than the value to be searched.
2. Accordingly, search is done to either half of the given list

Important Differences

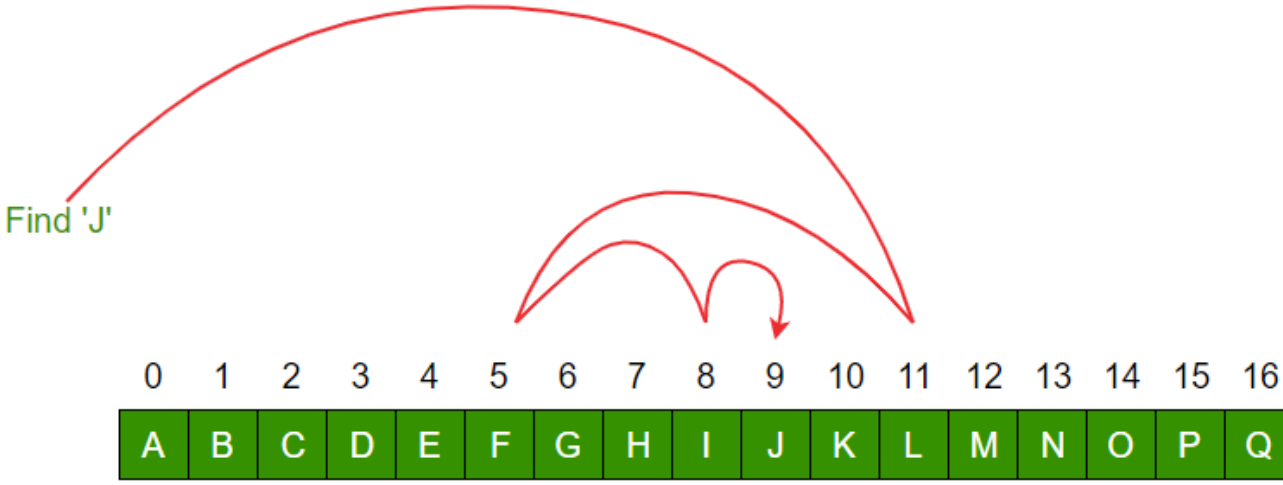
- Input data needs to be sorted in Binary Search and not in Linear Search
- Linear search does the sequential access whereas Binary search access data randomly.
- Time complexity of linear search -O(n) , Binary search has time complexity O(log n).
- Linear search performs equality comparisons and Binary search performs ordering comparisons

Let us look at an example to compare the two:

Linear Search to find the element “J” in a given sorted list from A-X



Binary Search to find the element "J" in a given sorted list from A-X



You may also see

- Searching and Sorting Articles
- Searching and Sorting Quizzes
- Practice Coding Questions

Source

https://www.geeksforgeeks.org/linear-search-vs-binary-search/

Chapter 142

Linear search using Multi-threading

Linear search using Multi-threading - GeeksforGeeks

Given a large file of integers, search for a particular element in it using multi-threading.

Examples:

```
Input : 1, 5, 7, 10, 12, 14, 15, 18, 20,
        22, 25, 27, 30, 64, 110, 220
Output :if key = 20
Key element found

Input :1, 5, 7, 10, 12, 14, 15, 18, 20,
       22, 25, 27, 30, 64, 110, 220
Output :if key = 202
Key not present
```

Prerequisite : Multi-threading

Approach :
First create n threads. Then, divide array in to four parts one section for each thread and apply linear search on individual section using multhreading and check whether the key element is present or not.

```
Command : g++ -pthread linear_thread.cpp
```

C++

```
 // CPP code to search for element in a
// very large file using Multithreading
#include <iostream>
#include <pthread.h>
using namespace std;

// Max size of array
#define max 16

// Max number of threads to create
#define thread_max 4

int a[max] = { 1, 5, 7, 10, 12, 14, 15,
               18, 20, 22, 25, 27, 30,
               64, 110, 220 };
int key = 202;

// Flag to indicate if key is found in a[]
// or not.
int f = 0;

int current_thread = 0;

// Linear search function which will
// run for all the threads
void* ThreadSearch(void* args)
{
    int num = current_thread++;

    for (int i = num * (max / 4);
         i < ((num + 1) * (max / 4)); i++)
    {
        if (a[i] == key)
            f = 1;
    }
}

// Driver Code
int main()
{
    pthread_t thread[thread_max];

    for (int i = 0; i < thread_max; i++) {
        pthread_create(&thread[i], NULL,
                       ThreadSearch, (void*)NULL);
    }

    for (int i = 0; i < thread_max; i++) {
```

```
        pthread_join(thread[i], NULL);
    }

    if (f == 1)
        cout << "Key element found" << endl;
    else
        cout << "Key not present" << endl;
    return 0;
}
```

Output:

```
Key not present
```

Exercise: The above code divides array into four subarrays. Extend this to take a parameter that decides number of divisions (or threads).

Source

https://www.geeksforgeeks.org/linear-search-using-multi-threading/

Chapter 143

Longest Palindromic Substring using Palindromic Tree | Set 3

Longest Palindromic Substring using Palindromic Tree | Set 3 - GeeksforGeeks

Given a string, find the longest substring which is palindrome. For example, if the given string is “forgeeksskeegfor”, the output should be “geeksskeeg”.

Prerequisite : Palindromic Tree | Longest Palindromic Substring

Structure of Palindromic Tree :
Palindromic Tree’s actual structure is close to directed graph. It is actually a merged structure of two Trees which share some common nodes(see the figure below for better understanding). Tree nodes store palindromic substrings of given string by storing their indices.
This tree consists of two types of edges :
1. Insertion edge (weighted edge)
2. Maximum Palindromic Suffix (un-weighted)

Insertion Edge :
Insertion edge from a node u to v with some weight x means that the node v is formed by inserting x at the front and end of the string at u. As ‘u’ is already a palindrome, hence the resulting string at node v will also be a palindrome. x will be a single character for every edge. Therefore, a node can have max 26 insertion edges (considering lower letter string).
Maximum Palindromic Suffix Edge :
As the name itself indicates that for a node this edge will point to its Maximum Palindromic Suffix String node. We will not be considering the complete string itself as the Maximum Palindromic Suffix as this will make no sense(self loops). For simplicity purpose, we will call it as Suffix edge(by which we mean maximum suffix except the complete string). It is quite obvious that every node will have only 1 Suffix Edge as we will not store duplicate strings in the tree.
We will create all the palindromic substrings and then return the last one we got, since that would be the longest palindromic substring so far.
Since a Palindromic Tree stores the palindromes in order of arrival of a certain character,

so the Longest will always be at the last index of our tree array.
Below is the implementation of above approach :

```
 // CPP code for Longest Palindromic substring
// using Palindromic Tree data structure
#include <bits/stdc++.h>
using namespace std;

#define MAXN  1000

struct Node
{
    // store start and end indexes of current
    // Node inclusively
    int start, end;

    // stores length of substring
    int length;

    // stores insertion Node for all characters a-z
    int insertionEdge[26];

    // stores the Maximum Palindromic Suffix Node for
    // the current Node
    int suffixEdge;
};

// two special dummy Nodes as explained above
Node root1, root2;

// stores Node information for constant time access
Node tree[MAXN];

// Keeps track the current Node while insertion
int currNode;
string s;
int ptr;

// Function to insert edge in tree
void insert(int currIndex)
{
    // Finding X, such that s[currIndex]
    // + X + s[currIndex] is palindrome.
    int temp = currNode;

    while (true)
    {
        int currLength = tree[temp].length;
```

```
    if (currIndex - currLength >= 1 &&
        (s[currIndex] == s[currIndex -
         currLength - 1]))
                    break;

    temp = tree[temp].suffixEdge;
}

// Check if s[currIndex] + X +
// s[currIndex] is already Present in tree.
if (tree[temp].insertionEdge[s[currIndex] -
    'a'] != 0)
{
    currNode =
    tree[temp].insertionEdge[s[currIndex]
               - 'a'];

    return;
}

// Else Create new node;
ptr++;

tree[temp].insertionEdge[s[currIndex]
           - 'a'] = ptr;

tree[ptr].end = currIndex;

tree[ptr].length = tree[temp].length + 2;

tree[ptr].start = tree[ptr].end -
                  tree[ptr].length + 1;

// Setting suffix edge for newly Created Node.

currNode = ptr;
temp = tree[temp].suffixEdge;

// Longest Palindromic suffix for a
// string of length 1 is a Null string.
if (tree[currNode].length == 1) {
    tree[currNode].suffixEdge = 2;
    return;
}

// Else
while (true) {
    int currLength = tree[temp].length;
```

```
        if (currIndex - currLength >= 1 &&
            (s[currIndex] ==
             s[currIndex - currLength - 1]))
                break;

        temp = tree[temp].suffixEdge;
    }

    tree[currNode].suffixEdge =
        tree[temp].insertionEdge[s[currIndex] - 'a'];
}

// Driver code
int main()
{
    // Imaginary root's suffix edge points to
    // itself, since for an imaginary string
    // of length = -1 has an imaginary suffix
    // string. Imaginary root.
    root1.length = -1;
    root1.suffixEdge = 1;

    // NULL root's suffix edge points to
    // Imaginary root, since for a string
    // of length = 0 has an imaginary suffix string.
    root2.length = 0;
    root2.suffixEdge = 1;

    tree[1] = root1;
    tree[2] = root2;

    ptr = 2;
    currNode = 1;
    s = "forgeeksskeegfor";
    for (int i = 0; i < s.size(); i++)
            insert(i);

    // last will be the index of our last substring
    int last = ptr;

    for (int i = tree[last].start;
         i <= tree[last].end; i++)
                cout << s[i];

    return 0;
}
```

Output:

```
geeksskeeg
```

Source

https://www.geeksforgeeks.org/longest-palindromic-substring-using-palindromic-tree-set-3/

Chapter 144

Longest Subarray with first element greater than or equal to Last element

Longest Subarray with first element greater than or equal to Last element - GeeksforGeeks

Given an array arr[0..n-1] of n integers, find the maximum length subarray such that its first element is greater than or equal to the last element of the subarray.

Examples:

```
Input : arr[] = {-5, -1, 7, 5, 1, -2}
Output : 5
Explanation : Subarray {-1, 7, 5, 1, -2} forms maximum
length subarray with its first element greater than last.

Input : arr[] = {1, 5, 7}
Output : 1
```

Naive approach is to use two nested loops for every possible starting and ending element of the subarray and if it satisfies the condition then update the answer accordingly.
Time Complexity – O(n^2)

A **Better approach** is to use Binary search. In this approach for each element in the array we maximise the length of the subarray ending at that element. Then we take the maximum of the all the lengths.

We will take another array which is used as our search space for choosing the starting element of. We will keep adding element in it as we traverse the array if the element is greater than all the previous elements present in the search space.
The key observation here is that we will add an element to our search space only when it

is greater than all the element of the search space because if the element is lower any other element then it can never be chosen as the first element of the maximum length subarray since we would have a better choice.
Since the search space is always sorted in increasing order, we just need compare the current element of the array with last element of the search space and if and only if its greater than the last element we add it to the search space otherwise not.

Below is the implementation of the above approach

C++

```
 // CPP program to find length of the longest
// array with first element smaller than or
// equal to last element.
#include <algorithm>
#include <iostream>
using namespace std;

// Search function for searching the first element of the
// subarray which is greater or equal to the last element(num)
int binarySearch(int* searchSpace, int s, int e, int num)
{
    int ans;

    while (s <= e) {
        int mid = (s + e) / 2;

        if (searchSpace[mid] >= num) {
            ans = mid;
            e = mid - 1;
        }
        else
            s = mid + 1;
    }

    return ans;
}

// Returns length of the longest array with first
// element smaller than the last element.
int longestSubArr(int* arr, int n)
{
    // Search space for the potential first elements.
    int searchSpace[n];

    // It will store the Indexes of the elements
    // of search space in the original array.
    int index[n];
```

```
    // Initially the search space is empty.
    int j = 0;
    int ans = 0;

    for (int i = 0; i < n; ++i) {

        // We will add an ith element in the search
        // space if the search space is empty or if
        // the ith element is greater than the last
        // element of the search space.
        if (j == 0 or searchSpace[j - 1] < arr[i]) {
            searchSpace[j] = arr[i];
            index[j] = i;
            j++;
        }

        // we will search for the index first element in the
        // search space and we will use it find the
        // index of it in the original array.
        int idx = binarySearch(searchSpace, 0, j - 1, arr[i]);

        // Update the answer if the length of the subarray is
        // greater than the previously calculated lengths.
        ans = max(ans, i - index[idx] + 1);
    }
    return ans;
}

int main(int argc, char const* argv[])
{
    int arr[] = { -5, -1, 7, 5, 1, -2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << longestSubArr(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java program to find length
// of the longest array with
// first element smaller than
// or equal to last element.
class GFG
{

// Search function for searching
// the first element of the
```

```
// subarray which is greater or
// equal to the last element(num)
static int binarySearch(int []searchSpace, int s,
                        int e, int num)
{
int ans = 0;

while (s <= e)
{
    int mid = (s + e) / 2;

    if (searchSpace[mid] >= num)
    {
        ans = mid;
        e = mid - 1;
    }
    else
        s = mid + 1;
}

return ans;
}

// Returns length of the longest
// array with first element smaller
// than the last element.
static int longestSubArr(int []arr, int n)
{
// Search space for the
// potential first elements.
int []searchSpace = new int[n];

// It will store the Indexes
// of the elements of search
// space in the original array.
int []index = new int[n];

// Initially the search
// space is empty.
int j = 0;
int ans = 0;

for (int i = 0; i < n; ++i)
{

    // We will add an ith element
    // in the search space if the
    // search space is empty or if
```

```
    // the ith element is greater
    // than the last element of
    // the search space.
    if (j == 0 || searchSpace[j - 1] < arr[i])
    {
        searchSpace[j] = arr[i];
        index[j] = i;
        j++;
    }

    // we will search for the index
    // first element in the search
    // space and we will use it find
    // the index of it in the original array.
    int idx = binarySearch(searchSpace, 0,
                           j - 1, arr[i]);

    // Update the answer if the
    // length of the subarray is
    // greater than the previously
    // calculated lengths.
    ans = Math.max(ans, i - index[idx] + 1);
}
return ans;
}

// Driver Code
public static void main(String[] args)
{
    int arr[] = { -5, -1, 7, 5, 1, -2 };
    int n = arr.length;
    System.out.println(longestSubArr(arr, n));
}
}

// This code is contributed
// by ChitraNayal
```

Python 3

```
 # Python 3 program to find
# length of the longest array
# with first element smaller
# than or equal to last element.

# Search function for searching
# the first element of the
# subarray which is greater or
```

```
# equal to the last element(num)
def binarySearch(searchSpace, s, e, num):

    while (s <= e):
        mid = (s + e) // 2

        if searchSpace[mid] >= num :
            ans = mid
            e = mid - 1

        else:
            s = mid + 1

    return ans

# Returns length of the longest
# array with first element
# smaller than the last element.
def longestSubArr(arr, n):

    # Search space for the
    # potential first elements.
    searchSpace = [None] * n

    # It will store the Indexes
    # of the elements of search
    # space in the original array.
    index = [None] * n

    # Initially the search
    # space is empty.
    j = 0
    ans = 0

    for i in range(n):

        # We will add an ith element
        # in the search space if the
        # search space is empty or if
        # the ith element is greater
        # than the last element of
        # the search space.
        if (j == 0 or searchSpace[j - 1] < arr[i]) :
            searchSpace[j] = arr[i]
            index[j] = i
            j += 1

        # we will search for the index
```

```
        # first element in the search
        # space and we will use it
        # find the index of it in the
        # original array.
        idx = binarySearch(searchSpace, 0,
                           j - 1, arr[i])

        # Update the answer if the length
        # of the subarray is greater than
        # the previously calculated lengths.
        ans = max(ans, i - index[idx] + 1)

    return ans

if __name__ == "__main__":
    arr = [ -5, -1, 7, 5, 1, -2 ]
    n = len(arr)
    print(longestSubArr(arr, n))

# This code is contributed
# by ChitraNayal
```

C#

```
 // C# program to find length
// of the longest array with
// first element smaller than
// or equal to last element.
using System;

class GFG
{

// Search function for searching
// the first element of the
// subarray which is greater or
// equal to the last element(num)
static int binarySearch(int[] searchSpace,
                        int s, int e, int num)
{
int ans = 0;

while (s <= e)
{
    int mid = (s + e) / 2;

    if (searchSpace[mid] >= num)
    {
```

```
        ans = mid;
        e = mid - 1;
    }
    else
        s = mid + 1;
}

return ans;
}

// Returns length of the longest
// array with first element smaller
// than the last element.
static int longestSubArr(int[] arr, int n)
{

// Search space for the
// potential first elements.
int[] searchSpace = new int[n];

// It will store the Indexes
// of the elements of search
// space in the original array.
int[] index = new int[n];

// Initially the search
// space is empty.
int j = 0;
int ans = 0;

for (int i = 0; i < n; ++i)
{

    // We will add an ith element
    // in the search space if the
    // search space is empty or if
    // the ith element is greater
    // than the last element of
    // the search space.
    if (j == 0 || searchSpace[j - 1] < arr[i])
    {
        searchSpace[j] = arr[i];
        index[j] = i;
        j++;
    }

    // we will search for the index
    // first element in the search
```

```
    // space and we will use it find the
    // index of it in the original array.
    int idx = binarySearch(searchSpace, 0,
                           j - 1, arr[i]);

    // Update the answer if the
    // length of the subarray is
    // greater than the previously
    // calculated lengths.
    ans = Math.Max(ans, i - index[idx] + 1);
}
return ans;
}

// Driver code
public static void Main()
{
    int[] arr = { -5, -1, 7, 5, 1, -2 };
    int n = arr.Length;
    Console.Write(longestSubArr(arr, n));
}
}

// This code is contributed
// by ChitraNayal
```

PHP

```
 <?php
// PHP program to find length
// of the longest array with
// first element smaller than
// or equal to last element.


// Search function for searching
// the first element of the
// subarray which is greater or
// equal to the last element(num)
function binarySearch($searchSpace,
                      $s, $e, $num)
{
    $ans = 0;
    while ($s <= $e)
    {
        $mid = ($s + $e) / 2;

        if ($searchSpace[$mid] >= $num)
```

```
        {
            $ans = $mid;
            $e = $mid - 1;
        }
        else
        {
            $s = $mid + 1;
        }
    }

    return $ans;
}

// Returns length of the longest
// array with first element smaller
// than the last element.
function longestSubArr(&$arr, $n)
{
    // Search space for the
    // potential first elements.

    // It will store the Indexes
    // of the elements of search
    // space in the original array.

    // Initially the search
    // space is empty.
    $j = 0;
    $ans = 0;
    for ($i = 0; $i < $n; ++$i)
    {

        // We will add an ith element
        // in the search space if the
        // search space is empty or if
        // the ith element is greater
        // than the last element of the
        // search space.
        if ($j == 0 or
            $searchSpace[$j - 1] < $arr[$i])
        {
            $searchSpace[$j] = $arr[$i];
            $index[$j] = $i;
            $j++;
        }


        // we will search for the index
```

```
        // first element in the search
        // space and we will use it find
        // the index of it in the original
        // array.
        $idx = binarySearch($searchSpace, 0,
                            $j - 1, $arr[$i]);

        // Update the answer if the length
        // of the subarray is greater than
        // the previously calculated lengths.
        $ans = max($ans, $i - $index[$idx] + 1);
    }
    return $ans;
}

// Driver code
$arr = array(-5, -1, 7, 5, 1, -2);
$n = sizeof($arr);
echo (longestSubArr($arr, $n)) ;

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output

```
5
```

*Time Complexity – (N * log N)*
Binary search function takes Log N time and the it will be called N times.

Improved By : Shivi_Aggarwal, ChitraNayal

Source

https://www.geeksforgeeks.org/longest-subarray-with-first-element-greater-than-or-equal-to-last-element/

Chapter 145

Longest prefix which is also suffix

Longest prefix which is also suffix - GeeksforGeeks

Given a string s, find length of the longest prefix which is also suffix. The prefix and suffix should not overlap.

Examples:

```
Input : aabcdaabc
Output : 4
The string "aabc" is the longest
prefix which is also suffix.

Input : abcab
Output : 2

Input : aaaa
Output : 2
```

Simple Solution : Since overlapping of prefix and suffix is not allowed, we break the string from middle and start matching left and right string. If they are equal return size of any one string else try for shorter lengths on both sides.

Below is a solution of above approach!

C++

```
 // CPP program to find length of the longest
// prefix which is also suffix
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to find largest prefix which is also a suffix
int largest_prefix_suffix(const std::string &str) {

  int n = str.length();

  if(n < 2) {
    return 0;
  }

  int len = 0;
  int i = n/2;

  while(i < n) {
    if(str[i] == str[len]) {
      ++len;
      ++i;
    } else {
      if(len == 0) { // no prefix
        ++i;
      } else { // search for shorter prefixes
        --len;
      }
    }
  }

  return len;

}

// Driver code
int main() {

string s = "blablabla";

cout << largest_prefix_suffix(s);

return 0;
}
```

Java

```
 // Java program to find length of the longest
// prefix which is also suffix
class GFG {

    static int longestPrefixSuffix(String s)
    {
```

```
    int n = s.length();

    if(n < 2) {
        return 0;
    }

    int len = 0;
    int i = n/2;

    while(i < n) {
        if(s.charAt(i) == s.charAt(len)) {
        ++len;
        ++i;
    }
    else
    {
      if(len == 0) { // no prefix
        ++i;
      }
      else
      {
        // search for shorter prefixes
        --len;
      }
    }
  }

  return len;

}

    // Driver code
    public static void main (String[] args)
    {
        String s = "blablabla";
        System.out.println(longestPrefixSuffix(s));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find length
# of the longest prefix which
# is also suffix

def longestPrefixSuffix(s) :
```

```
    n = len(s)

    for res in range(n // 2, 0, -1) :

        # Check for shorter lengths
        # of first half.
        prefix = s[0: res]
        suffix = s[n - res: n]

        if (prefix == suffix) :
            return res


    # if no prefix and suffix match
    # occurs
    return 0

s = "blablabla"
print(longestPrefixSuffix(s))

# This code is contributed by Nikita Tiwari.
```

C#

```
 // C# program to find length of the longest
// prefix which is also suffix
using System;

class GFG {

    static int longestPrefixSuffix(String s)
    {
        int n = s.Length;

        if(n < 2)
            return 0;

        int len = 0;
        int i = n / 2;

        while(i < n) {
            if(s[i] == s[len]) {
                ++len;
                ++i;
            }
            else {
                if(len == 0) {
```

```
                    // no prefix
                    ++i;
                }
                else {

                    // search for shorter prefixes
                    --len;
                }
            }
        }

        return len;
    }

    // Driver code
    public static void Main ()
    {
        String s = "blablabla";

        Console.WriteLine(longestPrefixSuffix(s));
    }
}

// This code is contributed by vt_m.
```

Output:

```
3
```

Efficient Solution : The idea is to use preprocessing algorithm of KMP search. In the preprocessing algorithm, we build lps array which stores following values.

> lps[i] = the longest proper prefix of pat[0..i]
> which is also a suffix of pat[0..i].

C++

```
 // Efficient CPP program to find length of
// the longest prefix which is also suffix
#include<bits/stdc++.h>
using namespace std;

// Returns length of the longest prefix
// which is also suffix and the two do
// not overlap. This function mainly is
```

```
// copy computeLPSArray() of in below post
// https://www.geeksforgeeks.org/searching-for-patterns-set-2-kmp-algorithm/
int longestPrefixSuffix(string s)
{
    int n = s.length();

    int lps[n];
    lps[0] = 0; // lps[0] is always 0

    // length of the previous
    // longest prefix suffix
    int len = 0;

    // the loop calculates lps[i]
    // for i = 1 to n-1
    int i = 1;
    while (i < n)
    {
        if (s[i] == s[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // (pat[i] != pat[len])
        {
            // This is tricky. Consider
            // the example. AAACAAAA
            // and i = 7. The idea is
            // similar to search step.
            if (len != 0)
            {
                len = lps[len-1];

                // Also, note that we do
                // not increment i here
            }
            else // if (len == 0)
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    int res = lps[n-1];

    // Since we are looking for
```

```
    // non overlapping parts.
    return (res > n/2)? n/2 : res;
}

// Driver program to test above function
int main()
{
    string s = "abcab";
    cout << longestPrefixSuffix(s);
    return 0;
}
```

Java

```
 // Efficient Java program to find length of
// the longest prefix which is also suffix

class GFG
{
    // Returns length of the longest prefix
    // which is also suffix and the two do
    // not overlap. This function mainly is
    // copy computeLPSArray() of in below post
    // https://www.geeksforgeeks.org/searching-
    // for-patterns-set-2-kmp-algorithm/
    static int longestPrefixSuffix(String s)
    {
        int n = s.length();

        int lps[] = new int[n];

        // lps[0] is always 0
        lps[0] = 0;

        // length of the previous
        // longest prefix suffix
        int len = 0;

        // the loop calculates lps[i]
        // for i = 1 to n-1
        int i = 1;
        while (i < n)
        {
            if (s.charAt(i) == s.charAt(len))
            {
                len++;
                lps[i] = len;
                i++;
```

```
            }

             // (pat[i] != pat[len])
            else
            {
                // This is tricky. Consider
                // the example. AAACAAAA
                // and i = 7. The idea is
                // similar to search step.
                if (len != 0)
                {
                    len = lps[len-1];

                    // Also, note that we do
                    // not increment i here
                }

                // if (len == 0)
                else
                {
                    lps[i] = 0;
                    i++;
                }
            }
        }

        int res = lps[n-1];

        // Since we are looking for
        // non overlapping parts.
        return (res > n/2)? n/2 : res;
    }

    // Driver program
    public static void main (String[] args)
    {
        String s = "abcab";
        System.out.println(longestPrefixSuffix(s));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Efficient Python 3 program
# to find length of
# the longest prefix
```

```
# which is also suffix

# Returns length of the longest prefix
# which is also suffix and the two do
# not overlap. This function mainly is
# copy computeLPSArray() of in below post
# https://www.geeksforgeeks.org/searching-for-patterns-set-2-kmp-algorithm/
def longestPrefixSuffix(s) :
    n = len(s)
    lps = [0] * n   # lps[0] is always 0

    # length of the previous
    # longest prefix suffix
    l = 0

    # the loop calculates lps[i]
    # for i = 1 to n-1
    i = 1
    while (i < n) :
        if (s[i] == s[l]) :
            l = l + 1
            lps[i] = l
            i = i + 1

        else :

            # (pat[i] != pat[len])
            # This is tricky. Consider
            # the example. AAACAAAA
            # and i = 7. The idea is
            # similar to search step.
            if (l != 0) :
                l = lps[l-1]

                # Also, note that we do
                # not increment i here

            else :

                # if (len == 0)
                lps[i] = 0
                i = i + 1

    res = lps[n-1]

    # Since we are looking for
    # non overlapping parts.
    if(res > n/2) :
```

```
        return n//2
    else :
        return res


# Driver program to test above function
s = "abcab"
print(longestPrefixSuffix(s))


# This code is contributed
# by Nikita Tiwari.
```

C#

```
 // Efficient C# program to find length of
// the longest prefix which is also suffix
using System;

class GFG {

    // Returns length of the longest prefix
    // which is also suffix and the two do
    // not overlap. This function mainly is
    // copy computeLPSArray() of in below post
    // https://www.geeksforgeeks.org/searching-
    // for-patterns-set-2-kmp-algorithm/
    static int longestPrefixSuffix(string s)
    {
        int n = s.Length;

        int []lps = new int[n];

        // lps[0] is always 0
        lps[0] = 0;

        // length of the previous
        // longest prefix suffix
        int len = 0;

        // the loop calculates lps[i]
        // for i = 1 to n-1
        int i = 1;
        while (i < n)
        {
            if (s[i] == s[len])
            {
                len++;
```

```
                lps[i] = len;
                i++;
            }

            // (pat[i] != pat[len])
            else
            {

                // This is tricky. Consider
                // the example. AAACAAAA
                // and i = 7. The idea is
                // similar to search step.
                if (len != 0)
                {
                    len = lps[len-1];

                    // Also, note that we do
                    // not increment i here
                }

                // if (len == 0)
                else
                {
                    lps[i] = 0;
                    i++;
                }
            }
        }

        int res = lps[n-1];

        // Since we are looking for
        // non overlapping parts.
        return (res > n/2) ? n/2 : res;
    }

    // Driver program
    public static void Main ()
    {
        string s = "abcab";

        Console.WriteLine(longestPrefixSuffix(s));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// Efficient PHP program to find length of
// the longest prefix which is also suffix

// Returns length of the longest prefix
// which is also suffix and the two do
// not overlap. This function mainly is
// copy computeLPSArray() of in below post
// https://www.geeksforgeeks.org/searching-for-patterns-set-2-kmp-algorithm/
function longestPrefixSuffix($s)
{
    $n = strlen($s);

    $lps[$n] = NULL;

    // lps[0] is always 0
    $lps[0] = 0;

    // length of the previous
    // longest prefix suffix
    $len = 0;

    // the loop calculates lps[i]
    // for i = 1 to n-1
    $i = 1;
    while ($i < $n)
    {
        if ($s[$i] == $s[$len])
        {
            $len++;
            $lps[$i] = $len;
            $i++;
        }

        // (pat[i] != pat[len])
        else
        {

            // This is tricky. Consider
            // the example. AAACAAAA
            // and i = 7. The idea is
            // similar to search step.
            if ($len != 0)
            {
                $len = $lps[$len-1];

                // Also, note that we do
                // not increment i here
```

```
            }

            // if (len == 0)
            else
            {
                $lps[$i] = 0;
                $i++;
            }
        }
    }

    $res = $lps[$n-1];

    // Since we are looking for
    // non overlapping parts.
    return ($res > $n/2)? $n/2 : $res;
}

    // Driver Code
    $s = "abcab";
    echo longestPrefixSuffix($s);

// This code is contributed by nitin mittal
?>
```

Output:

```
2
```

Please refer computeLPSArray() ofKMP search for explanation.

Time Complexity : O(n)
Auxiliary Space : O(n)

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/longest-prefix-also-suffix/

Chapter 146

Longest subarray having average greater than or equal to x

Longest subarray having average greater than or equal to x - GeeksforGeeks

Given an array of integers and an integer x. Find length of maximum size subarray having average of integers greater than or equal to x.

Examples:

```
Input : arr[] = {-2, 1, 6, -3}, x = 3
Output : 2
Longest subarray is {1, 6} having average
3.5 greater than x = 3.

Input : arr[] = {2, -3, 3, 2, 1}, x = 2
Output : 3
Longest subarray is {3, 2, 1} having
average 2 equal to x = 2.
```

A **simple** solution is to one by one consider each subarray and find its average. If average is greater than or equal to x, then compare length of subarray with maximum length found so far. Time complexity of this solution is $O(n^2)$.

An **efficient** solution is to use binary search and prefix sum. Let the required Longest subarray be arr[i..j] and its length is l = j-i+1. Thus, mathematically it can be written as:

(Σ (arr[i..j]) / l) x
==> (Σ (arr[i..j]) / l) – x 0
==> (Σ (arr[i..j]) – x*l) / l 0 …(1)

The equation (1) is equivalent to subtracting x from each element of subarray and then taking average of resultant subarray. So the resultant subarray can be obtained by subtracting x from each element of array. Let the updated subarray is arr1. Equation (1) can be further simplified as:

$$\Sigma \text{ (arr1[i..j])} / l \quad 0$$
$$==> \Sigma \text{ (arr1[i..j])} \quad 0 \ldots(2)$$

Equation (2) is simply Longest subarray having sum greater than or equal to zero. The Longest subarray having sum greater than or equal to zero can be found by method discussed in following article:
Longest subarray having sum greater than k.

The step wise algo is:
1. Subtract x from each element of array.
2. Find Longest subarray having sum greater than or equal to zero in updated array using prefix sum and binary search.

Below is the implementation of above approach:

```
 // CPP program to find Longest subarray
// having average greater than or equal
// to x.
#include <bits/stdc++.h>

using namespace std;

// Comparison function used to sort preSum vector.
bool compare(const pair<int, int>& a, const pair<int, int>& b)
{
    if (a.first == b.first)
        return a.second < b.second;

    return a.first < b.first;
}

// Function to find index in preSum vector upto which
// all prefix sum values are less than or equal to val.
int findInd(vector<pair<int, int> >& preSum, int n, int val)
{

    // Starting and ending index of search space.
    int l = 0;
    int h = n - 1;
    int mid;

    // To store required index value.
    int ans = -1;
```

```
    // If middle value is less than or equal to
    // val then index can lie in mid+1..n
    // else it lies in 0..mid-1.
    while (l <= h) {
        mid = (l + h) / 2;
        if (preSum[mid].first <= val) {
            ans = mid;
            l = mid + 1;
        }
        else
            h = mid - 1;
    }

    return ans;
}

// Function to find Longest subarray having average
// greater than or equal to x.
int LongestSub(int arr[], int n, int x)
{
    int i;

    // Update array by subtracting x from
    // each element.
    for (i = 0; i < n; i++)
        arr[i] -= x;

    // Length of Longest subarray.
    int maxlen = 0;

    // Vector to store pair of prefix sum
    // and corresponding ending index value.
    vector<pair<int, int> > preSum;

    // To store current value of prefix sum.
    int sum = 0;

    // To store minimum index value in range
    // 0..i of preSum vector.
    int minInd[n];

    // Insert values in preSum vector.
    for (i = 0; i < n; i++) {
        sum = sum + arr[i];
        preSum.push_back({ sum, i });
    }
```

```
    sort(preSum.begin(), preSum.end(), compare);

    // Update minInd array.
    minInd[0] = preSum[0].second;

    for (i = 1; i < n; i++) {
        minInd[i] = min(minInd[i - 1], preSum[i].second);
    }

    sum = 0;
    for (i = 0; i < n; i++) {
        sum = sum + arr[i];

        // If sum is greater than or equal to 0,
        // then answer is i+1.
        if (sum >= 0)
            maxlen = i + 1;

        // If sum is less than 0, then find if
        // there is a prefix array having sum
        // that needs to be added to current sum to
        // make its value greater than or equal to 0.
        // If yes, then compare length of updated
        // subarray with maximum length found so far.
        else {
            int ind = findInd(preSum, n, sum);
            if (ind != -1 && minInd[ind] < i)
                maxlen = max(maxlen, i - minInd[ind]);
        }
    }

    return maxlen;
}

// Driver code.
int main()
{
    int arr[] = { -2, 1, 6, -3 };
    int n = sizeof(arr) / sizeof(arr[0]);

    int x = 3;

    cout << LongestSub(arr, n, x);
    return 0;
}
```

Output:

2

Time Complexity: O(nlogn)
Auxiliary Space: O(n)

Another Approach :
Use approach of https://www.geeksforgeeks.org/largest-sum-contiguous-subarray/ just add a length variable and use the logic if the length is greater then previous saved length.

```
 // C program to find Longest subarray
// having average greater than or equal
// to x.
#include <stdio.h>

// Function to find Longest subarray having average
// greater than or equal to x.
int max_subarray(int arr[],int x,int n)
{
    int length=0 ,temp =0 ;
    int max=0,sum=0,i;
    for(i=0;i<n;i++)
    {
        sum = sum+arr[i];
        temp++;
        if(sum>max && sum/temp>=x && temp>=length)
        {
            max= sum ;
            length =temp;
        }
        if(sum<0)
        {
            sum=0;
            temp=0;
        }

    }

return length;
}

// Drive Code
int main() {
int arr[] = {-2, 1, 6, -3 };
int x = 2;
int n = sizeof(arr)/sizeof(int);
int length = max_subarray(arr,x,n);
printf("%d",length);
```

```
return 0;
}
```

Output:

```
2
```

Time Complexity: O(n)
Auxiliary Space: O(n)

Improved By : DivyaPurwar1

Source

https://www.geeksforgeeks.org/longest-subarray-having-average-greater-than-or-equal-to-x/

Chapter 147

Longest subarray such that the difference of max and min is at-most one

Longest subarray such that the difference of max and min is at-most one - GeeksforGeeks

Given an array of n numbers where the difference between each number and the previous one doesn't exceed one. Find the longest contiguous subarray such that the difference between the maximum and minimum number in the range doesn't exceed one.

Examples:

```
Input : {3, 3, 4, 4, 5, 6}
Output : 4
The longest subarray here is {3, 3, 4, 4}

Input : {7, 7, 7}
Output : 3
The longest subarray here is {7, 7, 7}

Input : {9, 8, 8, 9, 9, 10}
Output : 5
The longest subarray here is {9, 8, 8, 9, 9}
```

If the difference between the maximum and the minimum numbers in the sequence doesn't exceed one then the sequence consisted of only one or two consecutive numbers. The idea is to traverse the array and keep track of current element and previous element in current subarray. If we find an element which is not same as current or previous, we update current and previous. We also update result if required.

C++

```
 // C++ code to find longest
// subarray with difference
// between max and min as at-most 1.
#include<bits/stdc++.h>
using namespace std;

int longestSubarray(int input[],
                    int length)
{
    int prev = -1;
    int current, next;
    int prevCount = 0, currentCount = 1;

    // longest constant range length
    int longest = 1;

    // first number
    current = input[0];

    for (int i = 1; i < length; i++)
    {
        next = input[i];

        // If we see same number
        if (next == current)
        {
            currentCount++;
        }

        // If we see different number,
        // but same as previous.
        else if (next == prev)
        {
            prevCount += currentCount;
            prev = current;
            current = next;
            currentCount = 1;
        }

        // If number is neither same
        // as previous nor as current.
        else
        {
            longest = max(longest,
                          currentCount + prevCount);
            prev = current;
            prevCount = currentCount;
            current = next;
```

```
            currentCount = 1;
        }
    }

    return max(longest,
               currentCount + prevCount);
}

// Driver Code
int main()
{
    int input[] = { 5, 5, 6, 7, 6 };
    int n = sizeof(input) / sizeof(int);
    cout << longestSubarray(input, n);
    return 0;
}

// This code is contributed
// by Arnab Kundu
```

Java

```
 // Java code to find longest subarray with difference
// between max and min as at-most 1.
public class Demo {

    public static int longestSubarray(int[] input)
    {
        int prev = -1;
        int current, next;
        int prevCount = 0, currentCount = 1;

        // longest constant range length
        int longest = 1;

        // first number
        current = input[0];

        for (int i = 1; i < input.length; i++) {
            next = input[i];

            // If we see same number
            if (next == current) {
                currentCount++;
            }

            // If we see different number, but
            // same as previous.
```

```
            else if (next == prev) {
                prevCount += currentCount;
                prev = current;
                current = next;
                currentCount = 1;
            }

            // If number is neither same as previous
            // nor as current.
            else {
                longest = Math.max(longest, currentCount + prevCount);
                prev = current;
                prevCount = currentCount;
                current = next;
                currentCount = 1;
            }
        }

        return Math.max(longest, currentCount + prevCount);
    }

    public static void main(String[] args)
    {
        int[] input = { 5, 5, 6, 7, 6 };
        System.out.println(longestSubarray(input));
    }
}
```

Python 3

```
# Python 3 code to find longest
# subarray with difference
# between max and min as at-most 1.
def longestSubarray(input, length):

prev = -1
prevCount = 0
currentCount = 1

# longest constant range length
longest = 1

# first number
current = input[0]

for i in range(1, length):

next = input[i]

# If we see same number
if next == current :
currentCount += 1
```

If we see different number,
but same as previous.
elif next == prev :
prevCount += currentCount
prev = current
current = next
currentCount = 1

# If number is neither same
# as previous nor as current.
else:
longest = max(longest,
currentCount +
prevCount)
prev = current
prevCount = currentCount
current = next
currentCount = 1

return max(longest,
currentCount + prevCount)

# Driver Code
if ___name___ == "___main___":
input = [5, 5, 6, 7, 6]
n = len(input)
print(longestSubarray(input, n))

This code is contributed
by ChitraNayal

C#

```
 // C# code to find longest
// subarray with difference
// between max and min as
// at-most 1.
using System;

class GFG
{
public static int longestSubarray(int[] input)
{
    int prev = -1;
    int current, next;
    int prevCount = 0,
        currentCount = 1;

    // longest constant
    // range length
```

```
    int longest = 1;

    // first number
    current = input[0];

    for (int i = 1;
             i < input.Length; i++)
    {
        next = input[i];

        // If we see same number
        if (next == current)
        {
            currentCount++;
        }

        // If we see different number,
        // but same as previous.
        else if (next == prev)
        {
            prevCount += currentCount;
            prev = current;
            current = next;
            currentCount = 1;
        }

        // If number is neither
        // same as previous
        // nor as current.
        else
        {
            longest = Math.Max(longest,
                               currentCount +
                               prevCount);
            prev = current;
            prevCount = currentCount;
            current = next;
            currentCount = 1;
        }
    }

    return Math.Max(longest,
                    currentCount + prevCount);
}

// Driver Code
public static void Main(String[] args)
{
```

```
    int[] input = {5, 5, 6, 7, 6};
    Console.WriteLine(longestSubarray(input));
}
}

// This code is contributed
// by Kirti_Mangal
```

PHP

```
 <?php
// PHP code to find longest subarray
// with difference between max and
// min as at-most 1.
function longestSubarray($input, $length)
{
    $prev = -1;
    $prevCount = 0;
    $currentCount = 1;

    // longest constant range length
    $longest = 1;

    // first number
    $current = $input[0];

    for ($i = 1; $i < $length; $i++)
    {
        $next = $input[$i];

        // If we see same number
        if ($next == $current)
        {
            $currentCount++;
        }

        // If we see different number,
        // but same as previous.
        else if ($next == $prev)
        {
            $prevCount += $currentCount;
            $prev = $current;
            $current = $next;
            $currentCount = 1;
        }

        // If number is neither same
        // as previous nor as current.
```

```
        else
        {
            $longest = max($longest,
                           $currentCount +
                           $prevCount);
            $prev = $current;
            $prevCount = $currentCount;
            $current = $next;
            $currentCount = 1;
        }
    }

    return max($longest,
               $currentCount + $prevCount);
}

// Driver Code
$input = array( 5, 5, 6, 7, 6 );
echo(longestSubarray($input, count($input)));

// This code is contributed
// by Arnab Kundu
?>
```

Output:

```
3
```

Time Complexity: O(n) where n is the length of the input array.

Improved By : Kirti_Mangal, andrew1234, ChitraNayal

Source

https://www.geeksforgeeks.org/longest-subarray-such-that-the-difference-of-max-and-min-is-at-most-one/

Chapter 148

Longest substring having K distinct vowels

Longest substring having K distinct vowels - GeeksforGeeks

Given a string s we have to find the length of the longest substring of s which contain exactly K distinct vowels.
Note : Consider uppercase and lowercase characters as two different characters.

Examples:

> **Input :** s = “tHeracEBetwEEntheTwo”, k = 1
> **Output :** 14
> **Explanation :** Longest substring with only 1 vowel is “cEBetwEEntheTw”
> and its length is 14.
>
> **Input :** s = “artyebui”, k = 2
> **Output :** 6
> **Explanation :** Longest substring with only 2 vowel is “rtyebu”

Brute-Force Approach : For each substring, we check for the criteria for K distinct vowel and check the length. Finally, the largest length will be the result.

Efficient Approach : Here we maintain the count of vowels occurring in the substring. Till K is not zero, we count the distinct vowel occurring in the substring. As K becomes negative, we start deleting the first vowel of the substring we have found till that time, so that it may be possible that new substring(larger length) is possible afterward. As we delete the vowel we decrease its count so that in new substring may contain that vowel occurring in the later part of the string. And as K is 0 we get the length of the substring.

Below is the implementation of the above approach

C++

```
 // CPP program to find the longest substring
// with k distinct vowels.
#include <bits/stdc++.h>
using namespace std;

#define MAX 128

// Function to check whether a character is
// vowel or not
bool isVowel(char x)
{
    return (x == 'a' || x == 'e' || x == 'i' ||
            x == 'o' || x == 'u' || x == 'A' ||
            x == 'E' || x == 'I' || x == 'O' ||
            x == 'U');
}

int KDistinctVowel(char s[], int k)
{
    // length of string
    int n = strlen(s);

    // array for count of characters
    int c[MAX];
    memset(c, 0, sizeof(c));

    // Initialize result to be
    // negative
    int result = -1;

    for (int i = 0, j = -1; i < n; ++i) {

        int x = s[i];

        // If letter is vowel then we
        // increment its count value
        // and decrease the k value so
        // that if we again encounter the
        // same vowel character then we
        // don't consider it for our result
        if (isVowel(x)) {
            if (++c[x] == 1) {

                // Decrementing the K value
                --k;
            }
        }
```

```
        // Till k is 0 above if condition run
        // after that this while loop condition
        // also become active. Here what we have
        // done actually is that, if K is less
        // than 0 then we eliminate the first
        // vowel we have encountered till that
        // time . Now K is incremented and k
        // becomes 0. So, now we calculate the
        // length of substring from the present
        // index to the deleted index of vowel
        // which result in our results.
        while (k < 0) {

            x = s[++j];
            if (isVowel(x)) {

                // decresing the count
                // so that it may appear
                // in another substring
                // appearing after this
                // present substring
                if (--c[x] == 0) {

                    // incrementing the K value
                    ++k;
                }
            }
        }

        // Checking the maximum value
        // of the result by comparing
        // the length of substring
        // whenever K value is 0 means
        // K distinct vowel is present
        // in substring
        if (k == 0)
            result = max(result, i - j);
    }
    return result;
}

// Driver code
int main(void)
{
    char s[] = "tHeracEBetwEEntheTwo";
    int k = 1;
    cout << KDistinctVowel(s, k);
    return 0;
```

```
}
```

C#

```
 // C# program to find the longest substring
// with k distinct vowels.
using System;

class GFG {

    static int MAX = 128;

    // Function to check whether a character is
    // vowel or not
    static bool isVowel(char x)
    {
        return (x == 'a' || x == 'e' || x == 'i' ||
                x == 'o' || x == 'u' || x == 'A' ||
                x == 'E' || x == 'I' || x == 'O' ||
                x == 'U');
    }

    static int KDistinctVowel(string s, int k)
    {
        // length of string
        int n = s.Length;

        // array for count of characters
        int []c = new int[MAX];
        Array.Clear(c, 0, c.Length);

        // Initialize result to be
        // negative
        int result = -1;

        for (int i = 0, j = -1; i < n; ++i) {

            char x = s[i];

            // If letter is vowel then we
            // increment its count value
            // and decrease the k value so
            // that if we again encounter the
            // same vowel character then we
            // don't consider it for our result
            if (isVowel(x)) {
                if (++c[x] == 1) {
```

```
                // Decrementing the K value
                --k;
            }
        }

        // Till k is 0 above if condition run
        // after that this while loop condition
        // also become active. Here what we have
        // done actually is that, if K is less
        // than 0 then we eliminate the first
        // vowel we have encountered till that
        // time . Now K is incremented and k
        // becomes 0. So, now we calculate the
        // length of substring from the present
        // index to the deleted index of vowel
        // which result in our results.
        while (k < 0) {

            x = s[++j];
            if (isVowel(x)) {

                // decresing the count
                // so that it may appear
                // in another substring
                // appearing after this
                // present substring
                if (--c[x] == 0) {

                    // incrementing the K value
                    ++k;
                }
            }
        }

        // Checking the maximum value
        // of the result by comparing
        // the length of substring
        // whenever K value is 0 means
        // K distinct vowel is present
        // in substring
        if (k == 0) {
            result = Math.Max(result, i - j);
        }
    }
    return result;
}

// Driver code
```

```
    static void Main()
    {
        string s = "tHeracEBetwEEntheTwo";
        int k = 1;
        Console.Write(KDistinctVowel(s, k));
    }
}

// This code is contributed Manish Shaw
// (manishshaw1)
```

Output:

```
7
```

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/longest-substring-having-k-distinct-vowels/

Chapter 149

Make all array elements equal with minimum cost

Make all array elements equal with minimum cost - GeeksforGeeks

Given an array which contains integer values, we need to make all values of this array equal to some integer value with minimum cost where the cost of changing an array value x to y is abs(x-y).
Examples :

```
Input  : arr[] = [1, 100, 101]
Output : 100
We can change all its values to 100 with minimum cost,
|1 - 100| + |100 - 100| + |101 - 100| = 100

Input  : arr[] = [4, 6]
Output : 2
We can change all its values to 5 with minimum cost,
|4 - 5| + |5 - 6| = 2
```

This problem can be solved by observing the cost while changing the target equal value, i.e. we will see the change in cost when target equal value is changed. It can be observed that, as we increase the target equal value the total cost decreases up to a limit and then starts increasing i.e. the cost graph with respect to target equal value is of U-shape and as cost graph is in U-shape, the ternary search can be applied to this search space and our goal is to get that bottom most point of the curve which will represent the smallest cost. We will make smallest and largest value of the array as the limit of our search space and then we will keep skipping 1/3 part of the search space until we reach to the bottom most point of our U-curve.
Please see below code for better understanding,

C++

```
 // C/C++ program to find minimum cost to
// make all elements equal
#include <bits/stdc++.h>
using namespace std;

// Utility method to compute cost, when
// all values of array are made equal to X
int computeCost(int arr[], int N, int X)
{
    int cost = 0;
    for (int i = 0; i < N; i++)
        cost += abs(arr[i] - X);
    return cost;
}

// Method to find minimum cost to make all
// elements equal
int minCostToMakeElementEqual(int arr[], int N)
{
    int low, high;
    low = high = arr[0];

    // setting limits for ternary search by
    // smallest and largest element
    for (int i = 0; i < N; i++) {
        if (low > arr[i])
            low = arr[i];
        if (high < arr[i])
            high = arr[i];
    }

    /* loop until difference between low and high
       become less than 3, because after that
       mid1 and mid2 will start repeating
    */
    while ((high - low) > 2) {
        // mid1 and mid2 are representative array
        // equal values of search space
        int mid1 = low + (high - low) / 3;
        int mid2 = high - (high - low) / 3;

        int cost1 = computeCost(arr, N, mid1);
        int cost2 = computeCost(arr, N, mid2);

        // if mid2 point gives more total cost,
        // skip third part
        if (cost1 < cost2)
            high = mid2;
```

```
        // if mid1 point gives more total cost,
        // skip first part
        else
            low = mid1;
    }

    // computeCost gets optimum cost by sending
    // average of low and high as X
    return computeCost(arr, N, (low + high) / 2);
}

// Driver code to test above method
int main()
{
    int arr[] = { 1, 100, 101 };
    int N = sizeof(arr) / sizeof(int);
    cout << minCostToMakeElementEqual(arr, N);
    return 0;
}
```

Java

```
 // JAVA Code for Make all array elements
// equal with minimum cost
class GFG {

    // Utility method to compute cost, when
    // all values of array are made equal to X
    public static int computeCost(int arr[], int N,
                                  int X)
    {
        int cost = 0;
        for (int i = 0; i < N; i++)
            cost += Math.abs(arr[i] - X);
        return cost;
    }

    // Method to find minimum cost to make all
    // elements equal
    public static int minCostToMakeElementEqual(int arr[],
                                                int N)
    {
        int low, high;
        low = high = arr[0];

        // setting limits for ternary search by
        // smallest and largest element
```

```
        for (int i = 0; i < N; i++) {
            if (low > arr[i])
                low = arr[i];
            if (high < arr[i])
                high = arr[i];
        }

        /* loop until difference between low and high
           become less than 3, because after that
           mid1 and mid2 will start repeating
        */
        while ((high - low) > 2) {
            // mid1 and mid2 are representative array
            // equal values of search space
            int mid1 = low + (high - low) / 3;
            int mid2 = high - (high - low) / 3;

            int cost1 = computeCost(arr, N, mid1);
            int cost2 = computeCost(arr, N, mid2);

            // if mid2 point gives more total cost,
            // skip third part
            if (cost1 < cost2)
                high = mid2;

            // if mid1 point gives more total cost,
            // skip first part
            else
                low = mid1;
        }

        // computeCost gets optimum cost by sending
        // average of low and high as X
        return computeCost(arr, N, (low + high) / 2);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int arr[] = { 1, 100, 101 };
        int N = arr.length;
        System.out.println(minCostToMakeElementEqual(arr, N));
    }
}
// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python3 program to find minimum
# cost to make all elements equal

# Utility method to compute cost, when
# all values of array are made equal to X
def computeCost(arr, N, X):

    cost = 0
    for i in range(N):
        cost += abs(arr[i] - X)
    return cost


# Method to find minimum cost to
# make all elements equal
def minCostToMakeElementEqual(arr, N):

    low = high = arr[0]

    # Setting limits for ternary search
    # by smallest and largest element
    for i in range(N):

        if (low > arr[i]): low = arr[i]
        if (high < arr[i]): high = arr[i]


    # loop until difference between low and
    # high become less than 3, because after
    # that mid1 and mid2 will start repeating
    while ((high - low) > 2):

        # mid1 and mid2 are representative
        # array equal values of search space
        mid1 = low + (high - low) // 3
        mid2 = high - (high - low) // 3

        cost1 = computeCost(arr, N, mid1)
        cost2 = computeCost(arr, N, mid2)

        # if mid2 point gives more total
        # cost, skip third part
        if (cost1 < cost2):
            high = mid2

        # if mid1 point gives more total
        # cost, skip first part
        else:
```

```
            low = mid1


    # computeCost gets optimum cost by
    # sending average of low and high as X
    return computeCost(arr, N, (low + high) // 2)

# Driver code
arr = [1, 100, 101]
N = len(arr)
print(minCostToMakeElementEqual(arr, N))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# Code to Make all array elements
// equal with minimum cost
using System;

class GFG {

    // Utility method to compute cost, when
    // all values of array are made equal to X
    public static int computeCost(int[] arr, int N,
                                            int X)
    {
        int cost = 0;
        for (int i = 0; i < N; i++)
            cost += Math.Abs(arr[i] - X);
        return cost;
    }

    // Method to find minimum cost to
    // make all elements equal
    public static int minCostToMakeElementEqual(int[] arr,
                                                  int N)
    {
        int low, high;
        low = high = arr[0];

        // setting limits for ternary search by
        // smallest and largest element
        for (int i = 0; i < N; i++) {
            if (low > arr[i])
                low = arr[i];
            if (high < arr[i])
                high = arr[i];
```

```
        }

        /* loop until difference between low and high
        become less than 3, because after that
        mid1 and mid2 will start repeating
        */
        while ((high - low) > 2) {

            // mid1 and mid2 are representative array
            // equal values of search space
            int mid1 = low + (high - low) / 3;
            int mid2 = high - (high - low) / 3;

            int cost1 = computeCost(arr, N, mid1);
            int cost2 = computeCost(arr, N, mid2);

            // if mid2 point gives more total cost,
            // skip third part
            if (cost1 < cost2)
                high = mid2;

            // if mid1 point gives more total cost,
            // skip first part
            else
                low = mid1;
        }

        // computeCost gets optimum cost by sending
        // average of low and high as X
        return computeCost(arr, N, (low + high) / 2);
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int[] arr = { 1, 100, 101 };
        int N = arr.Length;
        Console.Write(minCostToMakeElementEqual(arr, N));
    }
}

// This code is contributed by nitin mittal
```

PHP

```
 <?php
// PHP program to find minimum cost
// to make all elements equal
```

```
// Utility method to compute cost,
// when all values of array are
// made equal to X
function computeCost($arr, $N, $X)
{
    $cost = 0;
    for ($i = 0; $i < $N; $i++)
        $cost += abs($arr[$i] - $X);
    return $cost;
}

// Method to find minimum cost
// to make all elements equal
function minCostToMakeElementEqual($arr, $N)
{
    $low; $high;
    $low = $high = $arr[0];

    // setting limits for ternary
    // search by smallest and
    // largest element
    for ($i = 0; $i < $N; $i++)
    {
        if ($low > $arr[$i])
            $low = $arr[$i];
        if ($high < $arr[$i])
            $high = $arr[$i];
    }

    /* loop until difference between
    low and high become less than 3,
    because after that mid1 and mid2
    will start repeating */
    while (($high - $low) > 2)
    {
        // mid1 and mid2 are representative
        // array equal values of search space
        $mid1 = $low + (floor($high - $low) / 3);
        $mid2 = $high - ($high - $low) / 3;

        $cost1 = computeCost($arr, $N, $mid1);
        $cost2 = computeCost($arr, $N, $mid2);

        // if mid2 point gives more total
        // cost, skip third part
        if ($cost1 < $cost2)
            $high = $mid2;
```

```
        // if mid1 point gives more
        // total cost, skip first part
        else
            $low = $mid1;
    }

    // computeCost gets optimum cost by
    // sending average of low and high as X
    return computeCost($arr, $N, ($low +
                                  $high) / 2);
}

// Driver Code
$arr = array( 1, 100, 101 );
$N = sizeof($arr) / sizeof($arr[0]);
echo minCostToMakeElementEqual($arr, $N);

// This code is contributed by nitin mittal.
?>
```

Output :

```
100
```

Time Complexity : O (n Log n)

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/make-array-elements-equal-minimum-cost/

Chapter 150

Making elements distinct in a sorted array by minimum increments

Making elements distinct in a sorted array by minimum increments - GeeksforGeeks

Given a sorted integer array. We need to make array elements distinct by increasing values and keeping array sum minimum possible. We need to print the minimum possible sum as output.

Examples:

```
Input : arr[] = { 2, 2, 3, 5, 6 } ;
Output : 20
Explanation : We make the array as {2,
3, 4, 5, 6}. Sum becomes 2 + 3 + 4 +
5 + 6 = 20

Input : arr[] = { 20, 20 } ;
Output : 41
Explanation : We make {20, 21}

Input :  arr[] = { 3, 4, 6, 8 };
Output : 21
Explanation : All elements are unique
so result is sum of each elements.
```

Method 1:
1. Traverse each element of array .
2. if arr[i] == arr[i-1] then update each element of array by adding 1 from i-th(current)

position to where element is either equal to its previous element or has become less than previous (because previous was increased).
3. After traversing of each element return sum .

C++

```
 // CPP program to make sorted array elements
// distinct by incrementing elements and keeping
// sum to minimum.
#include <iostream>
using namespace std;

// To find minimum sum of unique elements.
int minSum(int arr[], int n)
{
    int sum = arr[0];

    for (int i = 1; i < n; i++) {
        if (arr[i] == arr[i - 1]) {

            // While current element is same as
            // previous or has become smaller
            // than previous.
            int j = i;
            while (j < n && arr[j] <= arr[j - 1]) {
                arr[j] = arr[j] + 1;
                j++;
            }
        }
         sum = sum + arr[i];
     }

    return sum;
}

// Driver code
int main()
{
    int arr[] = { 2, 2, 3, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << minSum(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java program to make sorted
```

```
// array elements distinct by
// incrementing elements and
// keeping sum to minimum.
import java.io.*;

class GFG
{
    // To find minimum sum
    // of unique elements.
    static int minSum(int arr[], int n)
    {
        int sum = arr[0];

        for (int i = 1; i < n; i++)
        {
            if (arr[i] == arr[i - 1]) {

                // While current element is same as
                // previous or has become smaller
                // than previous.
                int j = i;
                while (j < n && arr[j] <= arr[j - 1])
                {
                    arr[j] = arr[j] + 1;
                    j++;
                }
            }
            sum = sum + arr[i];
        }

        return sum;
    }

    // Driver code
    public static void main (String[] args)
    {
        int arr[] = { 2, 2, 3, 5, 6 };
        int n = arr.length;
        System.out.println(minSum(arr, n));
    }
}

// This code is contributed by Ansu Kumari
```

Python3

```
 # Python3 program to make sorted array elements
# distinct by incrementing elements and keeping
```

```
# sum to minimum.

# To find minimum sum of unique elements.
def minSum(arr, n):
    sm = arr[0]

    for i in range(1, n):
        if arr[i] == arr[i - 1]:

            # While current element is same as
            # previous or has become smaller
            # than previous.
            j = i
            while j < n and arr[j] <= arr[j - 1]:
                arr[j] = arr[j] + 1
                j += 1

        sm = sm + arr[i]

    return sm

# Driver code
arr = [ 2, 2, 3, 5, 6 ]
n = len(arr)
print(minSum(arr, n))

# This code is contributed by Ansu Kumari
```

C#

```
 // C# program to make sorted
// array elements distinct by
// incrementing elements and
// keeping sum to minimum.
using System;

class GFG
{
    // To find minimum sum
    // of unique elements.
    static int minSum(int []arr, int n)
    {
        int sum = arr[0];

        for (int i = 1; i < n; i++)
        {
            if (arr[i] == arr[i - 1]) {
```

```
                // While current element is same as
                // previous or has become smaller
                // than previous.
                int j = i;
                while (j < n && arr[j] <= arr[j - 1])
                {
                    arr[j] = arr[j] + 1;
                    j++;
                }
            }
            sum = sum + arr[i];
        }

        return sum;
    }

    // Driver code
    public static void Main ()
    {
        int []arr = { 2, 2, 3, 5, 6 };
        int n = arr.Length;
        Console.WriteLine(minSum(arr, n));
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP program to make sorted array
// elements distinct by incrementing
// elements and keeping sum to minimum.

// To find minimum sum of unique
// elements.
function minSum($arr, $n)
{
    $sum = $arr[0];

    for ($i = 1; $i < $n; $i++) {
        if ($arr[$i] == $arr[$i - 1])
        {

            // While current element is
            // same as previous or has
            // become smaller than
            // previous.
```

```
            $j = $i;
            while ($j < $n && $arr[$j]
                        <= $arr[$j - 1])
            {
                $arr[$j] = $arr[$j] + 1;
                $j++;
            }
        }
        $sum = $sum + $arr[$i];
    }

    return $sum;
}

// Driver code
    $arr = array ( 2, 2, 3, 5, 6 );
    $n = sizeof($arr) ;
    echo minSum($arr, $n),"\n";

// This code is contributed by ajit
?>
```

Output :

```
20
```

Time Complexity : **O(n^2)**

Method 2:
1. Traverse each element of array .
2. if arr[i] <= prev then pdate prev by adding 1 and update sum by adding prev. Else update prev by cur element and update sum by adding cur element(arr[i]).
3. After traversing of each element return sum .

C++

```
 // Efficient CPP program to make sorted array
// elements distinct by incrementing elements
// and keeping sum to minimum.
#include <iostream>
using namespace std;

// To find minimum sum of unique elements.
int minSum(int arr[], int n)
{
    int sum = arr[0], prev = arr[0];
```

```
    for (int i = 1; i < n; i++) {

        // If violation happens, make current
        // value as 1 plus previous value and
        // add to sum.
        if (arr[i] <= prev) {
            prev = prev + 1;
            sum = sum + prev;
        }

        // No violation.
        else {
            sum = sum + arr[i];
            prev = arr[i];
        }
    }

    return sum;
}

// Drivers code
int main()
{
    int arr[] = { 2, 2, 3, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << minSum(arr, n) << endl;
    return 0;
}
```

Java

```
 // Efficient Java program to make sorted array
// elements distinct by incrementing elements
// and keeping sum to minimum.
import java.io.*;

class GFG {

    // To find minimum sum of unique elements.
    static int minSum(int arr[], int n)
    {
        int sum = arr[0], prev = arr[0];

        for (int i = 1; i < n; i++) {

            // If violation happens, make current
            // value as 1 plus previous value and
            // add to sum.
```

```
            if (arr[i] <= prev) {
                prev = prev + 1;
                sum = sum + prev;
            }

            // No violation.

            else {
                sum = sum + arr[i];
                prev = arr[i];
            }
        }

        return sum;
    }

    // Drivers code
    public static void main (String[] args) {

        int arr[] = { 2, 2, 3, 5, 6 };
        int n = arr.length;

        System.out.println(minSum(arr, n));
    }
}

// This code is contributed by Ansu Kumari.
```

Python3

```
 # Efficient Python program to make sorted array
# elements distinct by incrementing elements
# and keeping sum to minimum.

# To find minimum sum of unique elements
def minSum(arr, n):

    sum = arr[0]; prev = arr[0]

    for i in range(1, n):

        # If violation happens, make current
        # value as 1 plus previous value and
        # add to sum.
        if arr[i] <= prev:
            prev = prev + 1
            sum = sum + prev
```

```
        # No violation.
        else :
            sum = sum + arr[i]
            prev = arr[i]

    return sum

# Drivers code
arr = [ 2, 2, 3, 5, 6 ]
n = len(arr)
print(minSum(arr, n))

# This code is contributed by Ansu Kumari
```

C#

```
 // Efficient C# program to make sorted array
// elements distinct by incrementing elements
// and keeping sum to minimum.
using System;

class GFG {

    // To find minimum sum of unique elements.
    static int minSum(int []arr, int n)
    {
        int sum = arr[0], prev = arr[0];

        for (int i = 1; i < n; i++) {

            // If violation happens, make current
            // value as 1 plus previous value and
            // add to sum.
            if (arr[i] <= prev) {
                prev = prev + 1;
                sum = sum + prev;
            }

            // No violation.

            else {
                sum = sum + arr[i];
                prev = arr[i];
            }
        }

        return sum;
    }
```

```
    // Drivers code
    public static void Main () {

        int []arr = { 2, 2, 3, 5, 6 };
        int n = arr.Length;

        Console.WriteLine(minSum(arr, n));
    }
}

// This code is contributed by vt_m .
```

PHP

```
 <?php
// Efficient PHP program to
// make sorted array elements
// distinct by incrementing
// elements and keeping sum
// to minimum.

// To find minimum sum
// of unique elements.
function minSum($arr, $n)
{
    $sum = $arr[0];
    $prev = $arr[0];

    for ( $i = 1; $i < $n; $i++)
    {

        // If violation happens,
        // make current value as
        // 1 plus previous value
        // and add to sum.
        if ($arr[$i] <= $prev)
        {
            $prev = $prev + 1;
            $sum = $sum + $prev;
        }

        // No violation.
        else
        {
            $sum = $sum + $arr[$i];
            $prev = $arr[$i];
        }
```

```
    }

    return $sum;
}

// Driver code
$arr = array(2, 2, 3, 5, 6);
$n = count($arr);
echo minSum($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
20
```

Time Complexity: O(n)

Improved By : jit_t, vt_m

Source

https://www.geeksforgeeks.org/making-elements-distinct-sorted-array-minimum-increments/

Chapter 151

Maximum absolute difference of value and index sums

Maximum absolute difference of value and index sums - GeeksforGeeks

Given an unsorted array A of N integers, [illegible] Return maximum value of ***f(i, j)*** for all ***1 i, j N.***
f(i, j) or absolute difference of two elements of an array A is defined as $|\mathbf{A[i]} - \mathbf{A[j]}| + |\mathbf{i} - \mathbf{j}|$, where $|\boldsymbol{A}|$ denotes
the absolute value of A.

Examples:

```
We will calculate the value of f(i, j) for each pair
of (i, j) and return the maximum value thus obtained.

Input : A = {1, 3, -1}
Output : 5
f(1, 1) = f(2, 2) = f(3, 3) = 0
f(1, 2) = f(2, 1) = |1 - 3| + |1 - 2| = 3
f(1, 3) = f(3, 1) = |1 - (-1)| + |1 - 3| = 4
f(2, 3) = f(3, 2) = |3 - (-1)| + |2 - 3| = 5
So, we return 5.

Input : A = {3, -2, 5, -4}
Output : 10
f(1, 1) = f(2, 2) = f(3, 3) = f(4, 4) = 0
f(1, 2) = f(2, 1) = |3 - (-2)| + |1 - 2| = 6
f(1, 3) = f(3, 1) = |3 - 5| + |1 - 3| = 4
f(1, 4) = f(4, 1) = |3 - (-4)| + |1 - 4| = 10
f(2, 3) = f(3, 2) = |(-2) - 5| + |2 - 3| = 8
f(2, 4) = f(4, 2) = |(-2) - (-4)| + |2 - 4| = 4
```

```
f(3, 4) = f(4, 3) = |5 - (-4)| + |3 - 4| = 10

So, we return 10
```

A **naive brute force** approach is to calculate the value f(i, j) by iterating over all such pairs (i, j) and calculating the maximum absolute difference which is implemented below.

C++

```
 // Brute force C++ program to calculate the
// maximum absolute difference of an array.
#include <bits/stdc++.h>
using namespace std;

int calculateDiff(int i, int j, int arr[])
{
    // Utility function to calculate
    // the value of absolute difference
    // for the pair (i, j).
    return abs(arr[i] - arr[j]) + abs(i - j);
}

// Function to return maximum absolute
// difference in brute force.
int maxDistance(int arr[], int n)
{
    // Variable for storing the maximum
    // absolute distance throughout the
    // traversal of loops.
    int result = 0;

    // Iterate through all pairs.
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {

            // If the absolute difference of
            // current pair (i, j) is greater
            // than the maximum difference
            // calculated till now, update
            // the value of result.
            if (calculateDiff(i, j, arr) > result)
                result = calculateDiff(i, j, arr);
        }
    }
    return result;
}

// Driver program to test the above function.
```

```
int main()
{
    int arr[] = { -70, -64, -6, -56, 64,
                  61, -57, 16, 48, -98 };

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << maxDistance(arr, n) << endl;

    return 0;
}
```

Java

```
 // Java program to calculate the maximum
// absolute difference of an array.
public class MaximumAbsoluteDifference
{
    private static int calculateDiff(int i, int j,
                                     int[] array)
    {
        // Utility function to calculate
        // the value of absolute difference
        // for the pair (i, j).
        return Math.abs(array[i] - array[j]) +
                            Math.abs(i - j);
    }

    // Function to return maximum absolute
    // difference in brute force.
    private static int maxDistance(int[] array)
    {
        // Variable for storing the maximum
        // absolute distance throughout the
        // traversal of loops.
        int result = 0;

        // Iterate through all pairs.
        for (int i = 0; i < array.length; i++)
        {
            for (int j = i; j < array.length; j++)
            {

                // If the absolute difference of
                // current pair (i, j) is greater
                // than the maximum difference
                // calculated till now, update
                // the value of result.
```

```
                result = Math.max(result, calculateDiff(i, j, array));
            }
        }
        return result;
    }

    // Driver program to test above function
    public static void main(String[] args)
    {
        int[] array = { -70, -64, -6, -56, 64,
                        61, -57, 16, 48, -98 };
        System.out.println(maxDistance(array));
    }
}

// This code is contributed by Harikrishnan Rajan
```

Python3

```
 # Brute force Python 3 program
# to calculate the maximum
# absolute difference of an array.

def calculateDiff(i, j, arr):

    # Utility function to calculate
    # the value of absolute difference
    # for the pair (i, j).
    return abs(arr[i] - arr[j]) + abs(i - j)

# Function to return maximum
# absolute difference in
# brute force.
def maxDistance(arr, n):

    # Variable for storing the
    # maximum absolute distance
    # throughout the traversal
    # of loops.
    result = 0

    # Iterate through all pairs.
    for i in range(0,n):
        for j in range(i, n):

            # If the absolute difference of
            # current pair (i, j) is greater
            # than the maximum difference
```

```
            # calculated till now, update
            # the value of result.
            if (calculateDiff(i, j, arr) > result):
                result = calculateDiff(i, j, arr)

    return result

# Driver program
arr = [ -70, -64, -6, -56, 64,
         61, -57, 16, 48, -98 ]
n = len(arr)

print(maxDistance(arr, n))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to calculate the maximum
// absolute difference of an array.
using System;

public class MaximumAbsoluteDifference
{
    private static int calculateDiff(int i, int j,
                                     int[] array)
    {
        // Utility function to calculate
        // the value of absolute difference
        // for the pair (i, j).
        return Math.Abs(array[i] - array[j]) +
                            Math.Abs(i - j);
    }

    // Function to return maximum absolute
    // difference in brute force.
    private static int maxDistance(int[] array)
    {
        // Variable for storing the maximum
        // absolute distance throughout the
        // traversal of loops.
        int result = 0;

        // Iterate through all pairs.
        for (int i = 0; i < array.Length; i++)
        {
            for (int j = i; j < array.Length; j++)
            {
```

```
                // If the absolute difference of
                // current pair (i, j) is greater
                // than the maximum difference
                // calculated till now, update
                // the value of result.
                result = Math.Max(result, calculateDiff(i, j, array));
            }
        }
        return result;
    }

    // Driver program
    public static void Main()
    {
        int[] array = { -70, -64, -6, -56, 64,
                        61, -57, 16, 48, -98 };
        Console.WriteLine(maxDistance(array));
    }
}

// This code is contributed by vt_m
```

Output:

```
167
```

Time complexity: O(n^2)

An **efficient** solution in O(n) time complexity can be worked out using the properties of absolute values.
f(i, j) = |A[i] – A[j]| + |i – j| can be written in 4 ways (Since we are looking at max value, we don't even care if the value becomes negative as long as we are also covering the max value in some way).

```
Case 1: A[i] > A[j] and i > j
|A[i] - A[j]| = A[i] - A[j]
|i -j| = i - j
hence, f(i, j) = (A[i] + i) - (A[j] + j)

Case 2: A[i] < A[j] and i < j
|A[i] - A[j]| = -(A[i]) + A[j]
|i -j| = -(i) + j
hence, f(i, j) = -(A[i] + i) + (A[j] + j)

Case 3: A[i] > A[j] and i < j
|A[i] - A[j]| = A[i] - A[j]
```

```
|i -j| = -(i) + j
hence, f(i, j) = (A[i] - i) - (A[j] - j)

Case 4: A[i] < A[j] and i > j
|A[i] - A[j]| = -(A[i]) + A[j]
|i -j| = i - j
hence, f(i, j) = -(A[i] - i) + (A[j] - j)
```

Note that case 1 and 2 are equivalent and so are case 3 and 4 and hence we can design our algorithm only for two cases as it will cover all the possible cases.

1. Calculate the value of A[i] + i and A[i] – i for every element of the array while traversing through the array.

2. Then for the two equivalent cases, we find the maximum possible value. For that, we have to store minimum and maximum values of expressions A[i] + i and A[i] – i for all i.

3. Hence the required maximum absolute difference is maximum of two values i.e. max((A[i] + i) – (A[j] + j)) and max((A[i] – i) – (A[j] – j)). These values can be found easily in linear time.
   a. For max((A[i] + i) – (A[j] + j)) Maintain two variables max1 and min1 which will store maximum and minimum values of A[i] + i respectively. max((A[i] + i) – (A[j] + j)) = max1 – min1
   b. For max((A[i] – i) – (A[j] – j)). Maintain two variables max2 and min2 which will store maximum and minimum values of A[i] – i respectively. max((A[i] – i) – (A[j] – j)) = max2 – min2

Implementation using the above fast algorithm is given below.

C++

```
 // C++ program to calculate the maximum
// absolute difference of an array.
#include <bits/stdc++.h>
using namespace std;

// Function to return maximum absolue
// difference in linear time.
int maxDistance(int arr[], int n)
{
    // max and min variables as described
    // in algorithm.
    int max1 = INT_MIN, min1 = INT_MAX;
    int max2 = INT_MIN, min2 = INT_MAX;

    for (int i = 0; i < n; i++) {
```

```
        // Updating max and min variables
        // as described in algorithm.
        max1 = max(max1, arr[i] + i);
        min1 = min(min1, arr[i] + i);
        max2 = max(max2, arr[i] - i);
        min2 = min(min2, arr[i] - i);
    }

    // Calculating maximum absolute difference.
    return max(max1 - min1, max2 - min2);
}

// Driver program to test the above function.
int main()
{
    int arr[] = { -70, -64, -6, -56, 64,
                  61, -57, 16, 48, -98 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << maxDistance(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java program to calculate the maximum
// absolute difference of an array.
public class MaximumAbsoluteDifference
{
    // Function to return maximum absolue
    // difference in linear time.
    private static int maxDistance(int[] array)
    {
        // max and min variables as described
        // in algorithm.
        int max1 = Integer.MIN_VALUE;
        int min1 = Integer.MAX_VALUE;
        int max2 = Integer.MIN_VALUE;
        int `min2 = Integer.MAX_VALUE;

        for (int i = 0; i < array.length; i++)
        {

            // Updating max and min variables
            // as described in algorithm.
            max1 = Math.max(max1, array[i] + i);
            min1 = Math.min(min1, array[i] + i);
            max2 = Math.max(max2, array[i] - i);
            min2 = Math.min(min2, array[i] - i);
```

```
        }

        // Calculating maximum absolute difference.
        return Math.max(max1 - min1, max2 - min2);
    }

    // Driver program to test above function
    public static void main(String[] args)
    {
        int[] array = { -70, -64, -6, -56, 64,
                        61, -57, 16, 48, -98 };
        System.out.println(maxDistance(array));
    }
}

// This code is contributed by Harikrishnan Rajan
```

Python3

```
 # Python program to
# calculate the maximum
# absolute difference
# of an array.

# Function to return
# maximum absolue
# difference in linear time.
def maxDistance(array):

    # max and min variables as described
    # in algorithm.
    max1 = -2147483648
    min1 = +2147483647
    max2 = -2147483648
    min2 = +2147483647

    for i in range(len(array)):


        # Updating max and min variables
        # as described in algorithm.
        max1 = max(max1, array[i] + i)
        min1 = min(min1, array[i] + i)
        max2 = max(max2, array[i] - i)
        min2 = min(min2, array[i] - i)


    # Calculating maximum absolute difference.
```

```
    return max(max1 - min1, max2 - min2)


# Driver program to
# test above function

array = [ -70, -64, -6, -56, 64,
           61, -57, 16, 48, -98 ]

print(maxDistance(array))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to calculate the maximum
// absolute difference of an array.
using System;

public class MaximumAbsoluteDifference
{
    // Function to return maximum absolue
    // difference in linear time.
    private static int maxDistance(int[] array)
    {
        // max and min variables as described
        // in algorithm.
        int max1 = int.MinValue ;
        int min1 = int.MaxValue ;
        int max2 = int.MinValue ;
        int min2 =int.MaxValue ;

        for (int i = 0; i < array.Length; i++)
        {

            // Updating max and min variables
            // as described in algorithm.
            max1 = Math.Max(max1, array[i] + i);
            min1 = Math.Min(min1, array[i] + i);
            max2 = Math.Max(max2, array[i] - i);
            min2 = Math.Min(min2, array[i] - i);
        }

        // Calculating maximum absolute difference.
        return Math.Max(max1 - min1, max2 - min2);
    }
```

```
    // Driver program
    public static void Main()
    {
        int[] array = { -70, -64, -6, -56, 64,
                        61, -57, 16, 48, -98 };
        Console.WriteLine(maxDistance(array));
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP program to calculate the maximum
// absolute difference of an array.

// Function to return maximum absolue
// difference in linear time.
function maxDistance( $arr, $n)
{

    // max and min variables as
    // described in algorithm.
    $max1 = PHP_INT_MIN; $min1 =
                    PHP_INT_MAX;
    $max2 = PHP_INT_MIN;$min2 =
                    PHP_INT_MAX;

    for($i = 0; $i < $n; $i++)
    {

        // Updating max and min variables
        // as described in algorithm.
        $max1 = max($max1, $arr[$i] + $i);
        $min1 = min($min1, $arr[$i] + $i);
        $max2 = max($max2, $arr[$i] - $i);
        $min2 = min($min2, $arr[$i] - $i);
    }

    // Calculating maximum
    // absolute difference.
    return max($max1 - $min1,
               $max2 - $min2);
}

    // Driver Code
    $arr = array(-70, -64, -6, -56, 64,
```

```
                61, -57, 16, 48, -98);
    $n = count($arr);
    echo maxDistance($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
167
```

Time Complexity: O(n)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/maximum-absolute-difference-value-index-sums/

Chapter 152

Maximum adjacent difference in an array in its sorted form

Maximum adjacent difference in an array in its sorted form - GeeksforGeeks

Given an array, find the maximum difference between its two consecutive elements in its sorted form.

Examples:

```
Input : arr[] = {1, 10, 5}
Output : 5
Sorted array would be {1, 5, 10} and
maximum adjacent difference would be
10 - 5 = 5

Input : arr[] = {2, 4, 8, 11}
Output : 4
```

A **simple solution** is to first sort the array, then traverse it and keep track of maximum difference between adjacent elements. Time complexity of this

An **efficient solution** is based on idea of Pigeonhole sorting. We dont actually sort the array, we just have to fill the buckets and keep track of maximum and minimum value of each bucket. If we found an empty bucket, The maximum gap would be the difference of **maximum value in previous bucket – minimum value in next bucket**.

Below is the C++ code for above approach.

```
 // CPP program to find maximum adjacent difference
// between two adjacent after sorting.
#include <bits/stdc++.h>
using namespace std;
```

```
int maxSortedAdjacentDiff(int* arr, int n)
{
    // Find maximum and minimum in arr[]
    int maxVal = arr[0], minVal = arr[0];
    for (int i = 1; i < n; i++) {
        maxVal = max(maxVal, arr[i]);
        minVal = min(minVal, arr[i]);
    }

    // Arrays to store maximum and minimum values
    // in n-1 buckets of differences.
    int maxBucket[n - 1];
    int minBucket[n - 1];
    fill_n(maxBucket, n - 1, INT_MIN);
    fill_n(minBucket, n - 1, INT_MAX);

    // Expected gap for every bucket.
    float delta = (float)(maxVal - minVal) / (float)(n - 1);

    // Traversing through array elements and
    // filling in appropriate bucket if bucket
    // is empty. Else updating bucket values.
    for (int i = 0; i < n; i++) {
        if (arr[i] == maxVal || arr[i] == minVal)
            continue;

        // Finding index of bucket.
        int index = (float)(floor(arr[i] - minVal) / delta);

        // Filling/Updating maximum value of bucket
        if (maxBucket[index] == INT_MIN)
            maxBucket[index] = arr[i];
        else
            maxBucket[index] = max(maxBucket[index], arr[i]);

        // Filling/Updating minimum value of bucket
        if (minBucket[index] == INT_MAX)
            minBucket[index] = arr[i];
        else
            minBucket[index] = min(minBucket[index], arr[i]);
    }

    // Finding maximum difference between maximum value
    // of previous bucket minus minimum of current bucket.
    int prev_val = minVal;
    int max_gap = 0;
    for (int i = 0; i < n - 1; i++) {
```

```
        if (minBucket[i] == INT_MAX)
            continue;
        max_gap = max(max_gap, minBucket[i] - prev_val);
        prev_val = maxBucket[i];
    }
    max_gap = max(max_gap, maxVal - prev_val);

    return max_gap;
}

int main()
{
    int arr[] = { 1, 10, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << maxSortedAdjacentDiff(arr, n) << endl;
    return 0;
}
```

Output:

```
5
```

Time complexity : O(n)
Auxiliary Space : O(n)

Source

https://www.geeksforgeeks.org/maximum-adjacent-difference-array-sorted-form/

Chapter 153

Maximum and minimum of an array using minimum number of comparisons

Maximum and minimum of an array using minimum number of comparisons - GeeksforGeeks

Write a C function to return minimum and maximum in an array. You program should make minimum number of comparisons.

First of all, how do we return multiple values from a C function? We can do it either using structures or pointers.

We have created a structure named pair (which contains min and max) to return multiple values.

```
 struct pair
{
  int min;
  int max;
};
```

And the function declaration becomes: struct pair getMinMax(int arr[], int n) where arr[] is the array of size n whose minimum and maximum are needed.

METHOD 1 (Simple Linear Search)
Initialize values of min and max as minimum and maximum of the first two elements respectively. Starting from 3rd, compare each element with max and min, and change max and min accordingly (i.e., if the element is smaller than min then change min, else if the element is greater than max then change max, else ignore the element)

```
 /* structure is used to return two values from minMax() */
#include<stdio.h>
```

```
struct pair
{
  int min;
  int max;
};

struct pair getMinMax(int arr[], int n)
{
  struct pair minmax;
  int i;

  /*If there is only one element then return it as min and max both*/
  if (n == 1)
  {
     minmax.max = arr[0];
     minmax.min = arr[0];
     return minmax;
  }

  /* If there are more than one elements, then initialize min
      and max*/
  if (arr[0] > arr[1])
  {
      minmax.max = arr[0];
      minmax.min = arr[1];
  }
  else
  {
      minmax.max = arr[1];
      minmax.min = arr[0];
  }

  for (i = 2; i<n; i++)
  {
    if (arr[i] >  minmax.max)
      minmax.max = arr[i];

    else if (arr[i] <  minmax.min)
      minmax.min = arr[i];
  }

  return minmax;
}

/* Driver program to test above function */
int main()
{
  int arr[] = {1000, 11, 445, 1, 330, 3000};
```

```
  int arr_size = 6;
  struct pair minmax = getMinMax (arr, arr_size);
  printf("nMinimum element is %d", minmax.min);
  printf("nMaximum element is %d", minmax.max);
  getchar();
}
```

Time Complexity: O(n)

In this method, total number of comparisons is 1 + 2(n-2) in worst case and 1 + n – 2 in best case.
In the above implementation, worst case occurs when elements are sorted in descending order and best case occurs when elements are sorted in ascending order.

METHOD 2 (Tournament Method)
Divide the array into two parts and compare the maximums and minimums of the the two parts to get the maximum and the minimum of the the whole array.

```
Pair MaxMin(array, array_size)
   if array_size = 1
      return element as both max and min
   else if arry_size = 2
      one comparison to determine max and min
      return that pair
   else    /* array_size  > 2 */
      recur for max and min of left half
      recur for max and min of right half
      one comparison determines true max of the two candidates
      one comparison determines true min of the two candidates
      return the pair of max and min
```

Implementation

```
 /* structure is used to return two values from minMax() */
#include<stdio.h>
struct pair
{
  int min;
  int max;
};

struct pair getMinMax(int arr[], int low, int high)
{
  struct pair minmax, mml, mmr;
  int mid;

  /* If there is only on element */
```

```
  if (low == high)
  {
     minmax.max = arr[low];
     minmax.min = arr[low];
     return minmax;
  }

  /* If there are two elements */
  if (high == low + 1)
  {
     if (arr[low] > arr[high])
     {
        minmax.max = arr[low];
        minmax.min = arr[high];
     }
     else
     {
        minmax.max = arr[high];
        minmax.min = arr[low];
     }
     return minmax;
  }

  /* If there are more than 2 elements */
  mid = (low + high)/2;
  mml = getMinMax(arr, low, mid);
  mmr = getMinMax(arr, mid+1, high);

  /* compare minimums of two parts*/
  if (mml.min < mmr.min)
    minmax.min = mml.min;
  else
    minmax.min = mmr.min;

  /* compare maximums of two parts*/
  if (mml.max > mmr.max)
    minmax.max = mml.max;
  else
    minmax.max = mmr.max;

  return minmax;
}

/* Driver program to test above function */
int main()
{
  int arr[] = {1000, 11, 445, 1, 330, 3000};
  int arr_size = 6;
```

```
  struct pair minmax = getMinMax(arr, 0, arr_size-1);
  printf("nMinimum element is %d", minmax.min);
  printf("nMaximum element is %d", minmax.max);
  getchar();
}
```

Time Complexity: O(n)
Total number of comparisons: let number of comparisons be T(n). T(n) can be written as follows:
Algorithmic Paradigm: Divide and Conquer

```
  T(n) = T(floor(n/2)) + T(ceil(n/2)) + 2
  T(2) = 1
  T(1) = 0
```

If n is a power of 2, then we can write T(n) as:

```
   T(n) = 2T(n/2) + 2
```

After solving above recursion, we get

```
  T(n)  = 3n/2 -2
```

Thus, the approach does 3n/2 -2 comparisons if n is a power of 2. And it does more than 3n/2 -2 comparisons if n is not a power of 2.

METHOD 3 (Compare in Pairs)
If n is odd then initialize min and max as first element.
If n is even then initialize min and max as minimum and maximum of the first two elements respectively.
For rest of the elements, pick them in pairs and compare their
maximum and minimum with max and min respectively.

```
 #include<stdio.h>

/* structure is used to return two values from minMax() */
struct pair
{
  int min;
  int max;
};

struct pair getMinMax(int arr[], int n)
{
  struct pair minmax;
```

```
int i;

/* If array has even number of elements then
  initialize the first two elements as minimum and
  maximum */
if (n%2 == 0)
{
  if (arr[0] > arr[1])
  {
    minmax.max = arr[0];
    minmax.min = arr[1];
  }
  else
  {
    minmax.min = arr[0];
    minmax.max = arr[1];
  }
  i = 2;  /* set the startung index for loop */
}

 /* If array has odd number of elements then
  initialize the first element as minimum and
  maximum */
else
{
  minmax.min = arr[0];
  minmax.max = arr[0];
  i = 1;  /* set the startung index for loop */
}

/* In the while loop, pick elements in pair and
   compare the pair with max and min so far */
while (i < n-1)
{
  if (arr[i] > arr[i+1])
  {
    if(arr[i] > minmax.max)
      minmax.max = arr[i];
    if(arr[i+1] < minmax.min)
      minmax.min = arr[i+1];
  }
  else
  {
    if (arr[i+1] > minmax.max)
      minmax.max = arr[i+1];
    if (arr[i] < minmax.min)
      minmax.min = arr[i];
  }
```

```
    i += 2; /* Increment the index by 2 as two
               elements are processed in loop */
  }

  return minmax;
}

/* Driver program to test above function */
int main()
{
  int arr[] = {1000, 11, 445, 1, 330, 3000};
  int arr_size = 6;
  struct pair minmax = getMinMax (arr, arr_size);
  printf("nMinimum element is %d", minmax.min);
  printf("nMaximum element is %d", minmax.max);
  getchar();
}
```

Time Complexity: O(n)

Total number of comparisons: Different for even and odd n, see below:

```
       If n is odd:    3*(n-1)/2
       If n is even:   1 Initial comparison for initializing min and max,
                           and 3(n-2)/2 comparisons for rest of the elements
                     =  1 + 3*(n-2)/2 = 3n/2 -2
```

Second and third approaches make equal number of comparisons when n is a power of 2.

In general, method 3 seems to be the best.

Improved By : kamleshbhalui

Source

https://www.geeksforgeeks.org/maximum-and-minimum-in-an-array/

Chapter 154

Maximum difference between groups of size two

Maximum difference between groups of size two - GeeksforGeeks

Given an array of even number of elements, form groups of 2 using these array elements such that the difference between the group with highest sum and the one with lowest sum is maximum.

Note: An element can be a part of one group only and it has to be a part of at least 1 group.

Examples:

```
Input : arr[] = {1, 4, 9, 6}
Output : 10
Groups formed will be (1, 4) and (6, 9),
the difference between highest sum group
(6, 9) i.e 15 and lowest sum group (1, 4)
i.e 5 is 10.
```

```
Input : arr[] = {6, 7, 1, 11}
Output : 11
Groups formed will be (1, 6) and (7, 11),
the difference between highest sum group
(7, 11) i.e 18 and lowest sum group (1, 6)
i.e 7 is 11.
```

Simple Approach: We can solve this problem by making all possible combinations and checking each set of combination difference between the group with highest sum and with the lowest sum. A total of n*(n-1)/2 such groups would be formed (nC2).

Time Complexity: O(n^3), because it will take O(n^2) to generate groups and to check against each group n iterations will be needed thus overall it takes O(n^3) time.

Efficient Approach: We can use the greedy approach. Sort the whole array and our result is sum of last two elements minus sum of first two elements.

C++

```
 // CPP program to find minimum difference
// between groups of highest and lowest
// sums.
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

ll CalculateMax(ll arr[], int n)
{
    // Sorting the whole array.
    sort(arr, arr + n);

    int min_sum = arr[0] + arr[1];
    int max_sum = arr[n-1] + arr[n-2];

    return abs(max_sum - min_sum);
}

// Driver code
int main()
{
    ll arr[] = { 6, 7, 1, 11 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << CalculateMax(arr, n) << endl;
    return 0;
}
```

PHP

```
 <?php
// PHP program to find minimum
// difference between groups of
// highest and lowest sums.
function CalculateMax($arr, $n)
{
    // Sorting the whole array.
    sort($arr);

    $min_sum = $arr[0] +
               $arr[1];
```

```
    $max_sum = $arr[$n - 1] +
               $arr[$n - 2];

    return abs($max_sum -
               $min_sum);
}

// Driver code
$arr = array (6, 7, 1, 11 );
$n = sizeof($arr);
echo CalculateMax($arr, $n), "\n" ;

// This code is contributed by ajit
?>
```

Output:

```
11
```

Time Complexity: O (n * log n)

Further Optimization :
Instead of sorting, we can find maximum two and minimum two in linear time and reduce time complexity to O(n).

Improved By : jit_t

Source

https://www.geeksforgeeks.org/maximum-difference-groups-size-two/

Chapter 155

Maximum difference between two subsets of m elements

Maximum difference between two subsets of m elements - GeeksforGeeks

Given an array of n integers and a number m, find the maximum possible difference between two sets of m elements chosen from given array.

Examples:

```
Input : arr[] = 1 2 3 4 5
            m = 4
Output : 4
The maximum four elements are 2, 3,
4 and 5. The minimum four elements are
1, 2, 3 and 4. The difference between
two sums is (2 + 3 + 4 + 5) - (1 + 2
+ 3 + 4) = 4

Input : arr[] = 5 8 11 40 15
           m = 2
Output : 42
The difference is (40 + 15) - (5  + 8)
```

The idea is to first sort the array, then find sum of first m elements and sum of last m elements. Finally return difference between two sums.

CPP

```
 // C++ program to find difference
// between max and min sum of array
#include <algorithm>
```

```
#include <iostream>
using namespace std;

// utility function
int find_difference(int arr[], int n, int m)
{
    int max = 0, min = 0;

    // sort array
    sort(arr, arr + n);

    for (int i = 0, j = n - 1;
         i < m; i++, j--) {
        min += arr[i];
        max += arr[j];
    }

    return (max - min);
}

// Driver code
int main()
{
    int arr[] = { 1, 2, 3, 4, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int m = 4;
    cout << find_difference(arr, n, m);
    return 0;
}
```

Java

```
 // Java program to find difference
// between max and min sum of array
import java.util.Arrays;

class GFG {
    // utility function
    static int find_difference(int arr[], int n,
                               int m)
    {
        int max = 0, min = 0;

        // sort array
        Arrays.sort(arr);

        for (int i = 0, j = n - 1;
             i < m; i++, j--) {
```

```
            min += arr[i];
            max += arr[j];
        }

        return (max - min);
    }

    // Driver program
    public static void main(String arg[])
    {
        int arr[] = { 1, 2, 3, 4, 5 };
        int n = arr.length;
        int m = 4;
        System.out.print(find_difference(arr, n, m));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python program to
# find difference
# between max and
# min sum of array

def find_difference(arr, n, m):
    max = 0; min = 0

    # sort array
    arr.sort();
    j = n-1
    for i in range(m):
        min += arr[i]
        max += arr[j]
        j = j - 1

    return (max - min)

# Driver code
if __name__ == "__main__":
    arr = [1, 2, 3, 4, 5]
    n = len(arr)
    m = 4

    print(find_difference(arr, n, m))

# This code is contributed by
```

```
# Harshit Saini
```

C#

```
 // C# program to find difference
// between max and min sum of array
using System;

class GFG {

    // utility function
    static int find_difference(int[] arr, int n,
                                           int m)
    {
        int max = 0, min = 0;

        // sort array
        Array.Sort(arr);

        for (int i = 0, j = n - 1;
            i < m; i++, j--) {
            min += arr[i];
            max += arr[j];
        }

        return (max - min);
    }

    // Driver program
    public static void Main()
    {
        int[] arr = { 1, 2, 3, 4, 5 };
        int n = arr.Length;
        int m = 4;
        Console.Write(find_difference(arr, n, m));
    }
}

// This code is contributed by nitin mittal
```

PHP

```
 <?php
// PHP program to find difference
// between max and min sum of array

// utility function
```

```
function find_difference($arr, $n, $m)
{
    $max = 0; $min = 0;

    // sort array
    sort($arr);
    sort( $arr,$n);

    for($i = 0, $j = $n - 1; $i <$m; $i++, $j--)
    {
        $min += $arr[$i];
        $max += $arr[$j];
    }

    return ($max - $min);
}

// Driver code
{
    $arr = array(1, 2, 3, 4, 5);
    $n = sizeof($arr) / sizeof($arr[0]);
    $m = 4;
    echo find_difference($arr, $n, $m);
    return 0;
}

// This code is contributed by nitin mittal.
?>
```

Output:

```
4
```

We can optimize the above solution using more efficient approaches discussed in below post.
k largest(or smallest) elements in an array | added Min Heap method

Improved By : nitin mittal, SanyamAggarwal

Source

https://www.geeksforgeeks.org/difference-maximum-sum-minimum-sum-n-m-elementsin-review/

Chapter 156

Maximum element in a very large array using pthreads

Maximum element in a very large array using pthreads - GeeksforGeeks

Given very large array of integers, find maximum within the array using multhreading.

Examples:

```
Input :  1, 5, 7, 10, 12, 14, 15, 18, 20,
         22, 25, 27, 30, 64, 110, 220
Output :Maximun Element is : 220

Input :  10, 50, 70, 100, 120, 140, 150, 180,
         200, 220, 250, 270, 300, 640, 110, 220
Output : Maximun Element is : 640
```

Prerequisite : Multithreading

Note : Useful in large files of size MB/GB.

How to run :
It can only be run on linux envirenment.
command :

```
>> gcc -pthread maximum.c
>> ./a.out
```

```
 // C++ code to find maximum of an array using Multithreading
#include <pthread.h>
#include <stdio.h>
```

```
#include <stdlib.h>

// Size of array
#define max 16

// Max number of thread
#define Th_max 4

// Array
int a[max] = { 1, 5, 7, 10, 12, 14, 15, 18, 20,
               22, 25, 27, 300, 64, 110, 220 };

// Array to store max of threads
int max_num[Th_max] = { 0 };
int thread_no = 0;

// Function to find maximum
void maximum(void* arg)
{
    int i, num = thread_no++;
    int maxs = 0;

    for (i = num * (max / 4); i < (num + 1) * (max / 4); i++) {
        if (a[i] > maxs)
            maxs = a[i];
    }

    max_num[num] = maxs;
}

// Driver code
int main()
{
    int maxs = 0;
    int i;
    pthread_t threads[Th_max];

    // creating 4 threads
    for (i = 0; i < Th_max; i++)
        pthread_create(&threads[i], NULL,
                       maximum, (void*)NULL);

    // joining 4 threads i.e. waiting for
    // all 4 threads to complete
    for (i = 0; i < Th_max; i++)
        pthread_join(threads[i], NULL);

    // Finding max element in an array
```

```
    // by individual threads
    for (i = 0; i < Th_max; i++) {
        if (max_num[i] > maxs)
            maxs = max_num[i];
    }

    printf("Maximun Element is : %d", maxs);

    return 0;
}
```

Output:

```
Maximun Element is : 300
```

Source

https://www.geeksforgeeks.org/maximum-element-large-array-using-pthreads/

Chapter 157

Maximum elements that can be made equal with k updates

Maximum elements that can be made equal with k updates - GeeksforGeeks

Given an array and a value k. We have to find the maximum number of equal elements possible for the array so that we can increase the elements of the array by incrementing a total of at-most k.

Examples:

> **Input :** array = { 2, 4, 9 }, k = 3
> **Output :** 2
> We are allowed to do at most three increments. We can make two elements 4 by increasing 2 by 2. Note that we can not make two elements 9 as converting 4 to 9 requires 5 increments.
>
> **Input :** array = { 5, 5, 3, 1 }, k = 5
> **Output :** 3
> **Explanation:** Here 1st and 2nd elements are equal. Then we can increase 3rd element 3 upto 5. Then k becomes (k-2) = 3. Now we can't increase 1 to 5 because k value is 3 and we need 4 for the updation. Thus equal elements possible are 3. Here we can also increase 1 to 5. Then also we have 3 because we can't update 3 to 5.
>
> **Input :** array = { 5, 5, 3, 1 }, k = 6
> **Output :** 4

Naive Approach: In the naive approach we have an algorithm in **O(n^2)** time in which we check for each element how many other elements can be incremented so that they will become equal to them.

Efficient Approach: In this approach, first we will sort the array. Then we maintain two arrays. First is prefix sum array which stores the prefix sum of the array and another is maxx[] array which stores the maximum element found till every point, i.e., max[i] means

maximum element from 1 to i. After storing these values in prefix[] array and maxx[] array, we do the binary search from 1 to n(number of elements of the array) to calculate how many elements which can be incremented to make them equal. In the binary search, we use one function in which we determine *what is the number of elements can be incremented to make them equal to a single value.*

C++

```
 // C++ program to find maximum elements that can
// be made equal with k updates
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the maximum number of
// equal elements possible with atmost K increment
// of values .Here we have done sliding window
// to determine that whether there are x number of
// elements present which on increment will become
// equal. The loop here will run in fashion like
// 0...x-1, 1...x, 2...x+1, ...., n-x-1...n-1
bool ElementsCalculationFunc(int pre[], int maxx[],
                             int x, int k, int n)
{
    for (int i = 0, j = x; j <= n; j++, i++) {

        // It can be explained with the reasoning
        // that if for some x number of elements
        // we can update the values then the
        // increment to the segment (i to j having
        // length -> x) so that all will be equal is
        // (x*maxx[j]) this is the total sum of
        // segment and (pre[j]-pre[i]) is present sum
        // So difference of them should be less than k
        // if yes, then that segment length(x) can be
        // possible return true
        if (x * maxx[j] - (pre[j] - pre[i]) <= k)
            return true;
    }
    return false;
}

void MaxNumberOfElements(int a[], int n, int k)
{
    // sort the array in ascending order
    sort(a, a + n);
    int pre[n + 1]; // prefix sum array
    int maxx[n + 1]; // maximum value array
```

```
    // Initializing the prefix array
    // and maximum array
    for (int i = 0; i <= n; ++i) {
        pre[i] = 0;
        maxx[i] = 0;
    }

    // set the first element of both
    // array
    maxx[0] = a[0];
    pre[0] = a[0];
    for (int i = 1; i < n; i++) {

        // Calculating prefix sum of the array
        pre[i] = pre[i - 1] + a[i];

        // Calculating max value upto that position
        // in the array
        maxx[i] = max(maxx[i - 1], a[i]);
    }

    // Binary seach applied for
    // computation here
    int l = 1, r = n, ans;
    while (l < r) {
        int mid = (l + r) / 2;

        if (ElementsCalculationFunc(pre, maxx,
                                mid - 1, k, n)) {
            ans = mid;
            l = mid + 1;
        }
        else
            r = mid - 1;
    }

    // printing result
    cout << ans << "\n";
}

int main()
{
    int arr[] = { 2, 4, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    MaxNumberOfElements(arr, n, k);
    return 0;
}
```

Java

```
 // java program to find maximum elements that can
// be made equal with k updates

import java.util.Arrays;
public class GFG {

    // Function to calculate the maximum number of
    // equal elements possible with atmost K increment
    // of values .Here we have done sliding window
    // to determine that whether there are x number of
    // elements present which on increment will become
    // equal. The loop here will run in fashion like
    // 0...x-1, 1...x, 2...x+1, ...., n-x-1...n-1
    static boolean ElementsCalculationFunc(int pre[],
                      int maxx[], int x, int k, int n)
    {
        for (int i = 0, j = x; j <= n; j++, i++) {

            // It can be explained with the reasoning
            // that if for some x number of elements
            // we can update the values then the
            // increment to the segment (i to j having
            // length -> x) so that all will be equal is
            // (x*maxx[j]) this is the total sum of
            // segment and (pre[j]-pre[i]) is present sum
            // So difference of them should be less than k
            // if yes, then that segment length(x) can be
            // possible return true
            if (x * maxx[j] - (pre[j] - pre[i]) <= k)
                return true;
        }
        return false;
    }

    static void MaxNumberOfElements(int a[], int n, int k)
    {
        // sort the array in ascending order
        Arrays.sort(a);
        int []pre = new int[n + 1]; // prefix sum array
        int []maxx = new int[n + 1]; // maximum value array

        // Initializing the prefix array
        // and maximum array
        for (int i = 0; i <= n; ++i) {
            pre[i] = 0;
            maxx[i] = 0;
```

```
        }

        // set the first element of both
        // array
        maxx[0] = a[0];
        pre[0] = a[0];
        for (int i = 1; i < n; i++) {

            // Calculating prefix sum of the array
            pre[i] = pre[i - 1] + a[i];

            // Calculating max value upto that position
            // in the array
            maxx[i] = Math.max(maxx[i - 1], a[i]);
        }

        // Binary seach applied for
        // computation here
        int l = 1, r = n, ans=0;
        while (l < r) {

            int mid = (l + r) / 2;

            if (ElementsCalculationFunc(pre, maxx,
                                    mid - 1, k, n))
            {
                ans = mid;
                l = mid + 1;
            }
            else
                r = mid - 1;
        }

        // printing result
        System.out.print((int)ans + "\n");
    }

    public static void main(String args[]) {

        int arr[] = { 2, 4, 9 };
        int n = arr.length;
        int k = 3;

        MaxNumberOfElements(arr, n, k);

    }
}
```

```
// This code is contributed by Sam007
```

C#

```
 // C# program to find maximum elements that can
// be made equal with k updates
using System;

class GFG {

    // Function to calculate the maximum number of
    // equal elements possible with atmost K increment
    // of values .Here we have done sliding window
    // to determine that whether there are x number of
    // elements present which on increment will become
    // equal. The loop here will run in fashion like
    // 0...x-1, 1...x, 2...x+1, ...., n-x-1...n-1
    static bool ElementsCalculationFunc(int []pre,
                      int []maxx, int x, int k, int n)
    {
        for (int i = 0, j = x; j <= n; j++, i++) {

            // It can be explained with the reasoning
            // that if for some x number of elements
            // we can update the values then the
            // increment to the segment (i to j having
            // length -> x) so that all will be equal is
            // (x*maxx[j]) this is the total sum of
            // segment and (pre[j]-pre[i]) is present sum
            // So difference of them should be less than k
            // if yes, then that segment length(x) can be
            // possible return true
            if (x * maxx[j] - (pre[j] - pre[i]) <= k)
                return true;
        }
        return false;
    }

    static void MaxNumberOfElements(int []a, int n, int k)
    {
        // sort the array in ascending order
        Array.Sort(a);
        int []pre = new int[n + 1]; // prefix sum array
        int []maxx = new int[n + 1]; // maximum value array

        // Initializing the prefix array
        // and maximum array
        for (int i = 0; i <= n; ++i) {
```

```
        pre[i] = 0;
        maxx[i] = 0;
    }

    // set the first element of both
    // array
    maxx[0] = a[0];
    pre[0] = a[0];
    for (int i = 1; i < n; i++) {

        // Calculating prefix sum of the array
        pre[i] = pre[i - 1] + a[i];

        // Calculating max value upto that position
        // in the array
        maxx[i] = Math.Max(maxx[i - 1], a[i]);
    }

    // Binary seach applied for
    // computation here
    int l = 1, r = n, ans=0;
    while (l < r) {

        int mid = (l + r) / 2;

        if (ElementsCalculationFunc(pre, maxx,
                                mid - 1, k, n))
        {
            ans = mid;
            l = mid + 1;
        }
        else
            r = mid - 1;
    }

    // printing result
    Console.Write ((int)ans + "\n");
}

// Driver code
public static void Main()
{
    int []arr = { 2, 4, 9 };
    int n = arr.Length;
    int k = 3;

    MaxNumberOfElements(arr, n, k);
}
```

```
}

// This code is contributed by Sam007
```

Output:

```
2
```

Time Complexity : **O(nlog(n))**
Space Complexity : **O(n)**

Improved By : Sam007

Source

https://www.geeksforgeeks.org/maximum-elements-can-made-equal-k-updates/

Chapter 158

Maximum equlibrium sum in an array

Maximum equlibrium sum in an array - GeeksforGeeks

Given an array arr[]. Find maximum value of prefix sum which is also suffix sum for index i in arr[].

Examples :

```
Input : arr[] = {-1, 2, 3, 0, 3, 2, -1}
Output : 4
Prefix sum of arr[0..3] =
            Suffix sum of arr[3..6]

Input : arr[] = {-2, 5, 3, 1, 2, 6, -4, 2}
Output : 7
Prefix sum of arr[0..3] =
              Suffix sum of arr[3..7]
```

A **Simple Solution** is to one by one check the given condition (prefix sum equal to suffix sum) for every element and return the element that satisfies the given condition with maximum value.

C++

```
 // CPP program to find
// maximum equilibrium sum.
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to find
// maximum equilibrium sum.
int findMaxSum(int arr[], int n)
{
    int res = INT_MIN;
    for (int i = 0; i < n; i++)
    {
    int prefix_sum = arr[i];
    for (int j = 0; j < i; j++)
        prefix_sum += arr[j];

    int suffix_sum = arr[i];
    for (int j = n - 1; j > i; j--)
        suffix_sum += arr[j];

    if (prefix_sum == suffix_sum)
        res = max(res, prefix_sum);
    }
    return res;
}

// Driver Code
int main()
{
    int arr[] = {-2, 5, 3, 1,
                 2, 6, -4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findMaxSum(arr, n);
    return 0;
}
```

java

```
 // java program to find maximum
// equilibrium sum.
import java.io.*;

class GFG {

    // Function to find maximum
    // equilibrium sum.
    static int findMaxSum(int []arr, int n)
    {
        int res = Integer.MIN_VALUE;

        for (int i = 0; i < n; i++)
        {
            int prefix_sum = arr[i];
```

```
            for (int j = 0; j < i; j++)
                prefix_sum += arr[j];

            int suffix_sum = arr[i];

            for (int j = n - 1; j > i; j--)
                suffix_sum += arr[j];

            if (prefix_sum == suffix_sum)
                res = Math.max(res, prefix_sum);
        }

        return res;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int arr[] = {-2, 5, 3, 1, 2, 6, -4, 2 };
        int n = arr.length;
        System.out.println(findMaxSum(arr, n));
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // C# program to find maximum
// equilibrium sum.
using System;

class GFG {

    // Function to find maximum
    // equilibrium sum.
    static int findMaxSum(int []arr, int n)
    {
        int res = int.MinValue;

        for (int i = 0; i < n; i++)
        {
            int prefix_sum = arr[i];

            for (int j = 0; j < i; j++)
                prefix_sum += arr[j];
```

```
            int suffix_sum = arr[i];

            for (int j = n - 1; j > i; j--)
                suffix_sum += arr[j];

            if (prefix_sum == suffix_sum)
                res = Math.Max(res, prefix_sum);
        }

        return res;
    }

    // Driver Code
    public static void Main ()
    {
        int []arr = {-2, 5, 3, 1, 2, 6, -4, 2 };
        int n = arr.Length;
        Console.WriteLine(findMaxSum(arr, n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find
// maximum equilibrium sum.

// Function to find
// maximum equilibrium sum.
function findMaxSum( $arr, $n)
{
    $res = PHP_INT_MIN;
    for ( $i = 0; $i < $n; $i++)
    {
    $prefix_sum = $arr[$i];
    for ( $j = 0; $j < $i; $j++)
        $prefix_sum += $arr[$j];

    $suffix_sum = $arr[$i];
    for ( $j = $n - 1; $j > $i; $j--)
        $suffix_sum += $arr[$j];

    if ($prefix_sum == $suffix_sum)
        $res = max($res, $prefix_sum);
    }
    return $res;
```

```
}

// Driver Code
$arr = array(-2, 5, 3, 1,
             2, 6, -4, 2 );
$n = count($arr);
echo findMaxSum($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
7
```

Time Complexity: O(n^2)
Auxiliary Space: O(n)

A **Better Approach** is to traverse the array and store prefix sum for each index in array presum[], in which presum[i] stores sum of subarray arr[0..i]. Do another traversal of array and store suffix sum in another array suffsum[], in which suffsum[i] stores sum of subarray arr[i..n-1]. After this for each index check if presum[i] is equal to suffsum[i] and if they are equal then compare there value with overall maximum so far.

C++

```
 // CPP program to find
// maximum equilibrium sum.
#include <bits/stdc++.h>
using namespace std;

// Function to find maximum
// equilibrium sum.
int findMaxSum(int arr[], int n)
{
    // Array to store prefix sum.
    int preSum[n];

    // Array to store suffix sum.
    int suffSum[n];

    // Variable to store maximum sum.
    int ans = INT_MIN;

    // Calculate prefix sum.
    preSum[0] = arr[0];
    for (int i = 1; i < n; i++)
```

```
        preSum[i] = preSum[i - 1] + arr[i];

    // Calculate suffix sum and compare
    // it with prefix sum. Update ans
    // accordingly.
    suffSum[n - 1] = arr[n - 1];
    if (preSum[n - 1] == suffSum[n - 1])
        ans = max(ans, preSum[n - 1]);

    for (int i = n - 2; i >= 0; i--)
    {
        suffSum[i] = suffSum[i + 1] + arr[i];
        if (suffSum[i] == preSum[i])
            ans = max(ans, preSum[i]);
    }

    return ans;
}

// Driver Code
int main()
{
    int arr[] = { -2, 5, 3, 1,
                  2, 6, -4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findMaxSum(arr, n);
    return 0;
}
```

Java

```
 // Java program to find maximum equilibrium sum.
import java.io.*;

public class GFG {


    // Function to find maximum
    // equilibrium sum.
    static int findMaxSum(int []arr, int n)
    {

        // Array to store prefix sum.
        int []preSum = new int[n];

        // Array to store suffix sum.
        int []suffSum = new int[n];
```

```
        // Variable to store maximum sum.
        int ans = Integer.MIN_VALUE;

        // Calculate prefix sum.
        preSum[0] = arr[0];
        for (int i = 1; i < n; i++)
            preSum[i] = preSum[i - 1] + arr[i];

        // Calculate suffix sum and compare
        // it with prefix sum. Update ans
        // accordingly.
        suffSum[n - 1] = arr[n - 1];

        if (preSum[n - 1] == suffSum[n - 1])
            ans = Math.max(ans, preSum[n - 1]);

        for (int i = n - 2; i >= 0; i--)
        {
            suffSum[i] = suffSum[i + 1] + arr[i];

            if (suffSum[i] == preSum[i])
                ans = Math.max(ans, preSum[i]);
        }

        return ans;
    }

    // Driver Code
    static public void main (String[] args)
    {
        int []arr = { -2, 5, 3, 1, 2, 6, -4, 2 };
        int n = arr.length;

        System.out.println( findMaxSum(arr, n));
    }
}

// This code is contributed by anuj_67
```

C#

```
 // C# program to find maximum equilibrium sum.
using System;

public class GFG {


    // Function to find maximum
```

```
    // equilibrium sum.
    static int findMaxSum(int []arr, int n)
    {

        // Array to store prefix sum.
        int []preSum = new int[n];

        // Array to store suffix sum.
        int []suffSum = new int[n];

        // Variable to store maximum sum.
        int ans = int.MinValue;

        // Calculate prefix sum.
        preSum[0] = arr[0];
        for (int i = 1; i < n; i++)
            preSum[i] = preSum[i - 1] + arr[i];

        // Calculate suffix sum and compare
        // it with prefix sum. Update ans
        // accordingly.
        suffSum[n - 1] = arr[n - 1];

        if (preSum[n - 1] == suffSum[n - 1])
            ans = Math.Max(ans, preSum[n - 1]);

        for (int i = n - 2; i >= 0; i--)
        {
            suffSum[i] = suffSum[i + 1] + arr[i];

            if (suffSum[i] == preSum[i])
                ans = Math.Max(ans, preSum[i]);
        }

        return ans;
    }

    // Driver Code
    static public void Main ()
    {
        int []arr = { -2, 5, 3, 1, 2, 6, -4, 2 };
        int n = arr.Length;

        Console.WriteLine( findMaxSum(arr, n));
    }
}

// This code is contributed by anuj_67
```

Output:

7

Time Complexity: O(n)
Auxiliary Space: O(n)

Further Optimization :
We can avoid use of extra space by first computing total sum, then using it to find current prefix and suffix sums.

C++

```
 // CPP program to find
// maximum equilibrium sum.
#include <bits/stdc++.h>
using namespace std;

// Function to find
// maximum equilibrium sum.
int findMaxSum(int arr[], int n)
{
    int sum = accumulate(arr, arr + n, 0);
    int prefix_sum = 0, res = INT_MIN;
    for (int i = 0; i < n; i++)
    {
    prefix_sum += arr[i];
    if (prefix_sum == sum)
        res = max(res, prefix_sum);
    sum -= arr[i];
    }
    return res;
}

// Driver Code
int main()
{
    int arr[] = { -2, 5, 3, 1,
                  2, 6, -4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findMaxSum(arr, n);
    return 0;
}
```

Java

```
 // Java program to find maximum equilibrium
```

```
// sum.
import java.lang.Math.*;
import java.util.stream.*;

class GFG {

    // Function to find maximum equilibrium
    // sum.
    static int findMaxSum(int arr[], int n)
    {
        int sum = IntStream.of(arr).sum();
        int prefix_sum = 0,
        res = Integer.MIN_VALUE;

        for (int i = 0; i < n; i++)
        {
            prefix_sum += arr[i];

            if (prefix_sum == sum)
                res = Math.max(res, prefix_sum);
            sum -= arr[i];
        }

        return res;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int arr[] = { -2, 5, 3, 1,
                      2, 6, -4, 2 };
        int n = arr.length;
        System.out.print(findMaxSum(arr, n));
    }
}

// This code is contributed by Smitha.
```

C#

```
 // C# program to find maximum equilibrium sum.
using System.Linq;
using System;

class GFG {

    static int Add(int x, int y) {
        return x + y;
```

```
    }

    // Function to find maximum equilibrium
    // sum.
    static int findMaxSum(int []arr, int n)
    {
        int sum = arr.Aggregate(func:Add);
        int prefix_sum = 0,
        res = int.MinValue;

        for (int i = 0; i < n; i++)
        {
            prefix_sum += arr[i];

            if (prefix_sum == sum)
                res = Math.Max(res, prefix_sum);
            sum -= arr[i];
        }

        return res;
    }

    // Driver Code
    public static void Main()
    {
        int []arr = { -2, 5, 3, 1,
                      2, 6, -4, 2 };
        int n = arr.Length;
        Console.Write(findMaxSum(arr, n));
    }
}

// This code is contributed by Smitha.
```

Output :

```
7
```

Time Complexity: O(n)
Auxiliary Space: O(1)

Improved By : vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/maximum-equlibrium-sum-array/

Chapter 159

Maximum in an array that can make another array sorted

Maximum in an array that can make another array sorted - GeeksforGeeks

Given two arrays among which one is almost sorted with one element being in the wrong position making the array unsorted, the task is to swap that element with the maximum element from the second array which can be used to make the first array sorted.

Examples:

> Input: arr1 = {1, 3, 7, 4, 10},
> arr2 = {2, 1, 5, 8, 9}
> Output: 1 3 7 9 10
> Swap 4 with 9.
>
> Input: arr1 = {20, 1, 23},
> arr2 = {50, 26, 7}
> Output: Not Possible

Approach:
1. Get the index of the element which is making the array unsorted.
2. Get the maximum element from the second array satisfying the neighboring conditions of the element with wrong index i.e
(i) max element >= arr[wrong index-1]
(ii)max element <= arr[wrong index+1] if wrong index+1 exists

```
 // C++ program to make array sorted
#include <bits/stdc++.h>
using namespace std;

// Function to check whether there is any
// swappable element present to make the first
```

```
// array sorted
bool swapElement(int arr1[], int arr2[], int n)
{

    // wrongIdx is the index of the element
    // which is making the first array unsorted
    int wrongIdx = 0;
    for (int i = 1; i < n; i++) {
        if (arr1[i] < arr1[i - 1])
            wrongIdx = i;

    int maximum = INT_MIN;
    int maxIdx = -1;
    bool res = false;

    // Find the maximum element which satisfies the
    // the above mentioned neighboring conditions
    for (int i = 0; i < n; i++) {
        if (arr2[i] > maximum && arr2[i] >= arr1[wrongIdx - 1]) {
            if (wrongIdx + 1 <= n - 1 &&
                arr2[i] <= arr1[wrongIdx + 1]) {
                maximum = arr2[i];
                maxIdx = i;
                res = true;
            }
        }
    }

    // if res is true then swap the element
    // and make the first array sorted
    if (res)
        swap(arr1[wrongIdx], arr2[maxIdx]);

    return res;
}

// Function to print the sorted array if elements
// are swapped.
void getSortedArray(int arr1[], int arr2[], int n)
{
    if (swapElement(arr1, arr2, n))
        for (int i = 0; i < n; i++)
            cout << arr1[i] << " ";
    else
        cout << "Not Possible" << endl;
}

// Drivers code
```

```
int main()
{

    int arr1[] = { 1, 3, 7, 4, 10 };
    int arr2[] = { 2, 1, 6, 8, 9 };

    int n = sizeof(arr1) / sizeof(arr1[0]);

    getSortedArray(arr1, arr2, n);
}
```

Output:

```
1 3 7 9 10
```

Source

https://www.geeksforgeeks.org/maximum-in-an-array-that-can-make-another-array-sorted/

Chapter 160

Maximum in array which is at-least twice of other elements

Maximum in array which is at-least twice of other elements - GeeksforGeeks

Given an array of integers of length n. Our task is to return the index of the max element if the it is at least twice as much as every other number in the array. If the max element does not satisfy the condition return -1.

Examples:

```
Input : arr = {3, 6, 1, 0}
Output : 1
Here, 6 is the largest integer, and for
every other number in the array x, 6 is
more than twice as big as x. The index of
value 6 is 1, so we return 1.

Input : arr = {1, 2, 3, 4}
Output : -1
4 isn't at least as big as twice the value
of 3, so we return -1.
```

Approach : Scan through the array to find the unique largest element **m**, keeping track of it's index **maxIndex**. Scan through the array again. If we find some **x != m** with **m < 2*x**, we should return **-1**. Otherwise, we should return **maxIndex**.

C++

```
 // CPP program for Maximum of
// the array which is at least
// twice of other elements of
```

```
// the array.
#include<iostream>
using namespace std;

// Function to find the
// index of Max element
// that satisfies the
// condition
int findIndex(int arr[], int len) {

    // Finding index of
    // max of the array
    int maxIndex = 0;
    for (int i = 0; i < len; ++i)
        if (arr[i] > arr[maxIndex])
            maxIndex = i;

    // Returns -1 if the
    // max element is not
    // twice of the i-th
    // element.
    for (int i = 0; i < len; ++i)
        if (maxIndex != i &&
            arr[maxIndex] < 2 * arr[i])
            return -1;

    return maxIndex;
}

// Driver function
int main(){
    int arr[] = {3, 6, 1, 0};
    int len = sizeof(arr) / sizeof(arr[0]);

    cout<<(findIndex(arr, len));
}

// This code is contributed by Smitha Dinesh Semwal
```

Java

```
 // Java program for Maximum of the array
// which is at least twice of other elements
// of the array.
import java.util.*;
import java.lang.*;

class GfG {
```

```
    // Function to find the index of Max element
    // that satisfies the condition
    public static int findIndex(int[] arr) {

        // Finding index of max of the array
        int maxIndex = 0;
        for (int i = 0; i < arr.length; ++i)
            if (arr[i] > arr[maxIndex])
                maxIndex = i;

        // Returns -1 if the max element is not
        // twice of the i-th element.
        for (int i = 0; i < arr.length; ++i)
            if (maxIndex != i && arr[maxIndex] < 2 * arr[i])
                return -1;

        return maxIndex;
    }

    // Driver function
    public static void main(String argc[]){
        int[] arr = new int[]{3, 6, 1, 0};
        System.out.println(findIndex(arr));
    }
}
```

Python3

```
 # Python 3 program for Maximum of
# the array which is at least twice
# of other elements of the array.

# Function to find the index of Max
# element that satisfies the condition
def findIndex(arr):

    # Finding index of max of the array
    maxIndex = 0
    for i in range(0,len(arr)):
        if (arr[i] > arr[maxIndex]):
            maxIndex = i

    # Returns -1 if the max element is not
    # twice of the i-th element.
    for i in range(0,len(arr)):
        if (maxIndex != i and
                arr[maxIndex] < (2 * arr[i])):
```

```
            return -1

    return maxIndex


# Driver code
arr = [3, 6, 1, 0]
print(findIndex(arr))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program for Maximum of the array
// which is at least twice of other elements
// of the array.
using System;

class GfG {

    // Function to find the index of Max element
    // that satisfies the condition
    public static int findIndex(int[] arr) {

        // Finding index of max of the array
        int maxIndex = 0;
        for (int i = 0; i < arr.Length; ++i)
            if (arr[i] > arr[maxIndex])
                maxIndex = i;

        // Returns -1 if the max element is not
        // twice of the i-th element.
        for (int i = 0; i < arr.Length; ++i)
            if (maxIndex != i && arr[maxIndex] < 2 * arr[i])
                return -1;

        return maxIndex;
    }

    // Driver function
    public static void Main()
    {
        int[] arr = new int[]{3, 6, 1, 0};
        Console.WriteLine(findIndex(arr));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program for Maximum of
// the array which is at least
// twice of other elements of
// the array.

// Function to find the
// index of Max element
// that satisfies the
// condition
function findIndex($arr, $len)
{

    // Finding index of
    // max of the array
    $maxIndex = 0;
    for ( $i = 0; $i < $len; ++$i)
        if ($arr[$i] > $arr[$maxIndex])
            $maxIndex = $i;

    // Returns -1 if the
    // max element is not
    // twice of the i-th
    // element.
    for ($i = 0; $i < $len; ++$i)
        if ($maxIndex != $i and
            $arr[$maxIndex] < 2 * $arr[$i])
            return -1;

    return $maxIndex;
}

// Driver Code
$arr = array(3, 6, 1, 0);
$len = count($arr);

echo findIndex($arr, $len);

// This code is contributed by anuj_67.
?>
```

Output:

```
1
```

Time Complexity: $O(n)$

Improved By : vt_m

Source

https://www.geeksforgeeks.org/maximum-array-least-twice-elements/

Chapter 161

Maximum occurring character in a linked list

Maximum occurring character in a linked list - GeeksforGeeks

Given a linked list of characters. The task is to find the maximum occurring character in the linked list. if their are multiple answer return the first maximum occurring character.

Examples:

```
Input  : g -> e -> e -> k -> s
Output : e

Input  : a -> a -> b -> b -> c -> c -> d -> d
Output : d
```

Method 1:
Iteratively count the frequency of each character in a string and return the one which has maximum occurrence.

```
 // CPP program to count the maximum
// occurring character in linked list
#include <bits/stdc++.h>
using namespace std;

/* Link list node */
struct Node {
  char data;
  struct Node *next;
};

char maxChar(struct Node *head) {
```

```
  struct Node *p = head;

  int max = -1;
  char res;

  while (p != NULL) {

    // counting the frequency of current element
    // p->data
    struct Node *q = p->next;
    int count = 1;
    while (q != NULL) {
      if (p->data == q->data)
        count++;

      q = q->next;
    }

    // if current counting is greater than max
    if (max < count) {
      res = p->data;
      max = count;
    }

    p = p->next;
  }

  return res;
}

/* Push a node to linked list. Note that
   this function changes the head */
void push(struct Node **head_ref, char new_data) {
  struct Node *new_node = new Node;
  new_node->data = new_data;
  new_node->next = (*head_ref);
  (*head_ref) = new_node;
}

/* Drier program to test above function*/
int main() {
  /* Start with the empty list */
  struct Node *head = NULL;
  char str[] = "skeegforskeeg";
  int i;

  // this will create a linked list of
  // character "geeksforgeeks"
```

```
  for (i = 0; str[i] != '\0'; i++)
    push(&head, str[i]);

  cout << maxChar(head);

  return 0;
}
```

Output:

```
e
```

Time complexityO(N*N)

Method 2: (use count array)
Create a count array and count each character frequency return the maximum occurring character.

```
 // CPP program to count the maximum
// occurring character in linked list
#include <bits/stdc++.h>
using namespace std;

/* Link list node */
struct Node {
  char data;
  struct Node *next;
};

char maxChar(struct Node *head) {
  struct Node *p = head;
  int hash[256] = {0};

  // Storing element's frequencies
  // in a hash table.
  while (p != NULL) {
    hash[p->data]++;
    p = p->next;
  }

  p = head;

  int max = -1;
  char res;

  // calculating the first maximum element
  while (p != NULL) {
```

```
    if (max < hash[p->data]) {
      res = p->data;
      max = hash[p->data];
    }
    p = p->next;
  }
  return res;
}

/* Push a node to linked list. Note that
   this function changes the head */
void push(struct Node **head_ref, char new_data) {
  struct Node *new_node = new Node;
  new_node->data = new_data;
  new_node->next = (*head_ref);
  (*head_ref) = new_node;
}

/* Drier program to test above function*/
int main() {
  struct Node *head = NULL;
  char str[] = "skeegforskeeg";
  for (int i = 0; str[i] != '\0'; i++)
    push(&head, str[i]);
  cout << maxChar(head);
  return 0;
}
```

Output:

```
e
```

Time complexityO(N)

Source

https://www.geeksforgeeks.org/maximum-occurring-character-linked-list/

Chapter 162

Maximum product quadruple (sub-sequence of size 4) in array

Maximum product quadruple (sub-sequence of size 4) in array - GeeksforGeeks

Given an integer array, find a maximum product of a quadruple in the array.

Examples:

```
Input:  [10, 3, 5, 6, 20]
Output: 6000
Multiplication of 10, 5, 6 and 20

Input:  [-10, -3, -5, -6, -20]
Output: 6000

Input:  [1, -4, 3, -6, 7, 0]
Output: 504
```

Approach 1 (Naive, O(n^4) time, O(1) Space)
A simple solution is to check for every quadruple using four nested loops. Below is its implementation-

C++

```
 // A C++ program to find a maximum product of a
// quadruple in array of integers
#include <bits/stdc++.h>
using namespace std;
```

```
/* Function to find a maximum product of a
   quadruple in array of integers of size n */
int maxProduct(int arr[], int n)
{
    // if size is less than 4, no quadruple exists
    if (n < 4)
        return -1;

    // will contain max product
    int max_product = INT_MIN;

    for (int i = 0; i < n - 3; i++)
        for (int j = i + 1; j < n - 2; j++)
            for (int k = j + 1; k < n - 1; k++)
                for (int l = k + 1; l < n; l++)
                    max_product = max(max_product,
                   arr[i] * arr[j] * arr[k] * arr[l]);

    return max_product;
}

// Driver program to test above functions
int main()
{
    int arr[] = { 10, 3, 5, 6, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int max = maxProduct(arr, n);
    if (max == -1)
        cout << "No Quadruple Exists";
    else
        cout << "Maximum product is " << max;
    return 0;
}
```

Java

```
 // A Java program to find
// a maximum product of a
// quadruple in array of
// integers
import java.io.*;

class GFG
{

/* Function to find a
maximum product of a
quadruple in array of
```

```
integers of size n */
static int maxProduct(int arr[],
                      int n)
{
    // if size is less than 4,
    // no quadruple exists
    if (n < 4)
        return -1;

    // will contain
    // max product
    int max_product = Integer.MIN_VALUE;

    for (int i = 0;
             i < n - 3; i++)
        for (int j = i + 1;
                 j < n - 2; j++)
            for (int k = j + 1;
                     k < n - 1; k++)
                for (int l = k + 1;
                         l < n; l++)
                    max_product = Math.max(max_product,
                                           arr[i] * arr[j] *
                                           arr[k] * arr[l]);

    return max_product;
}

// Driver Code
public static void main (String[] args)
{
    int arr[] = { 10, 3, 5, 6, 20 };
    int n = arr.length;
    int max = maxProduct(arr, n);
    if (max == -1)
        System.out.println("No Quadruple Exists");
    else
        System.out.println("Maximum product is " + max);
}
}

// This code is contributed
// by anuj_67
```

Python3

```
 # Python3 program to find a
# maximum product of a
```

```
# quadruple in array of
# integers
import sys

# Function to find a maximum
# product of a quadruple in
# array of integers of size n
def maxProduct(arr, n):

    # if size is less than
    # 4, no quadruple exists
    if (n < 4):
        return -1;

    # will contain max product
    max_product = -sys.maxsize;

    for i in range(n - 3):
        for j in range(i + 1, n - 2):
            for k in range(j + 1, n - 1):
                for l in range(k + 1, n):
                    max_product = max(max_product,
                                      arr[i] * arr[j] *
                                      arr[k] * arr[l]);

    return max_product;

# Driver Code
arr = [10, 3, 5, 6, 20];
n = len(arr);
max = maxProduct(arr, n);

if (max == -1):
    print("No Quadruple Exists");
else:
    print("Maximum product is", max);

# This code is contributed
# by rahul
```

C#

```
 // A C# program to find
// a maximum product of a
// quadruple in array of
// integers
using System;
```

```
class GFG
{

/* Function to find a
maximum product of a
quadruple in array of
integers of size n */
static int maxProduct(int []arr,
                      int n)
{
    // if size is less than 4,
    // no quadruple exists
    if (n < 4)
        return -1;

    // will contain
    // max product
    int max_product = int.MinValue;

    for (int i = 0;
             i < n - 3; i++)
        for (int j = i + 1;
                 j < n - 2; j++)
            for (int k = j + 1;
                     k < n - 1; k++)
                for (int l = k + 1;
                         l < n; l++)
                    max_product = Math.Max(max_product,
                                           arr[i] * arr[j] *
                                           arr[k] * arr[l]);

    return max_product;
}

// Driver Code
public static void Main ()
{
    int []arr = {10, 3, 5, 6, 20};
    int n = arr.Length;
    int max = maxProduct(arr, n);
    if (max == -1)
        Console.WriteLine("No Quadruple Exists");
    else
        Console.WriteLine("Maximum product is " + max);
}
}

// This code is contributed
```

```
// by anuj_67
```

PHP

```
 <?php
// PHP program to find a
// maximum product of a
// quadruple in array of
// integers

// Function to find a maximum
// product of a quadruple in
// array of integers of size n
function maxProduct($arr, $n)
{
    // if size is less than
    // 4, no quadruple exists
    if ($n < 4)
        return -1;

    // will contain max product
    $max_product = PHP_INT_MIN;

    for ($i = 0; $i < $n - 3; $i++)
        for ($j = $i + 1; $j < $n - 2; $j++)
            for ($k = $j + 1; $k < $n - 1; $k++)
                for ($l = $k + 1; $l < $n; $l++)
                    $max_product = max($max_product,
                                       $arr[$i] * $arr[$j] *
                                       $arr[$k] * $arr[$l]);

    return $max_product;
}

// Driver Code
$arr = array(10, 3, 5, 6, 20);
$n = count($arr);
$max = maxProduct($arr, $n);
if ($max == -1)
    echo "No Quadruple Exists";
else
    echo "Maximum product is " , $max;

// This code is contributed
// by anuj_67
?>
```

Output :

```
Maximum product is 6000
```

Approach 2: O(nlogn) Time, O(1) Space

1. Sort the array using some efficient in-place sorting algorithm in ascending order.
2. Let x be the product of last four elements.
3. Let y be the product of first four elements.
4. Let z be the product of first two elements and last two elements.
5. Return the maximum of x, y and z.

Below is its implementation–

C++

```
 // A C++ program to find a maximum product of a
// quadruple in array of integers
#include <bits/stdc++.h>
using namespace std;

/* Function to find a maximum product of a quadruple
   in array of integers of size n */
int maxProduct(int arr[], int n)
{
    // if size is less than 4, no quadruple exists
    if (n < 4)
        return -1;

    // Sort the array in ascending order
    sort(arr, arr + n);

    int x = arr[n - 1] * arr[n - 2] * arr[n - 3] * arr[n - 4];
    int y = arr[0] * arr[1] * arr[2] * arr[3];
    int z = arr[0] * arr[1] * arr[n - 1] * arr[n - 2];

    // Return the maximum of x, y and z
    return max(x, max(y, z));
}

// Driver program to test above functions
int main()
{
    int arr[] = { -10, -3, 5, 6, -20 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int max = maxProduct(arr, n);
    if (max == -1)
        cout << "No Quadruple Exists";
    else
```

```
        cout << "Maximum product is " << max;

    return 0;
}
```

Java

```
 // A Java program to find a
// maximum product of a
// quadruple in array of integers
import java.io.*;
import java.util.Arrays;

class GFG
{

/* Function to find a
maximum product of a quadruple
in array of integers of size n */
static int maxProduct(int arr[],
                      int n)
{
    // if size is less than 4,
    // no quadruple exists
    if (n < 4)
        return -1;

    // Sort the array
    // in ascending order
    Arrays.sort(arr);

    int x = arr[n - 1] * arr[n - 2] *
            arr[n - 3] * arr[n - 4];
    int y = arr[0] * arr[1] *
            arr[2] * arr[3];
    int z = arr[0] * arr[1] *
            arr[n - 1] * arr[n - 2];

    // Return the maximum
    // of x, y and z
    return Math.max(x, Math.max(y, z));
}

// Driver Code
public static void main (String[] args)
{
    int arr[] = {-10, -3, 5, 6, -20};
    int n = arr.length;
```

```
    int max = maxProduct(arr, n);
    if (max == -1)
        System.out.println("No Quadruple Exists");
    else
        System.out.println("Maximum product is " +
                                                max);
}
}

// This code is contributed
// by anuj_67
```

C#

```
 // A C# program to find a
// maximum product of a
// quadruple in array of
// integers
using System;

class GFG
{

/* Function to find a
maximum product of a
quadruple in array of
integers of size n */
static int maxProduct(int []arr,
                      int n)
{
    // if size is less than 4,
    // no quadruple exists
    if (n < 4)
        return -1;

    // Sort the array
    // in ascending order
    Array.Sort(arr);

    int x = arr[n - 1] * arr[n - 2] *
            arr[n - 3] * arr[n - 4];
    int y = arr[0] * arr[1] *
            arr[2] * arr[3];
    int z = arr[0] * arr[1] *
            arr[n - 1] * arr[n - 2];

    // Return the maximum
    // of x, y and z
```

```
    return Math.Max(x, Math.Max(y, z));
}

// Driver Code
public static void Main ()
{
    int []arr = {-10, -3, 5, 6, -20};
    int n = arr.Length;
    int max = maxProduct(arr, n);
    if (max == -1)
        Console.WriteLine("No Quadruple Exists");
    else
        Console.WriteLine("Maximum product is " +
                                            max);

}
}

// This code is contributed
// by anuj_67
```

PHP

```
 <?php
// A PHP program to find a
// maximum product of a
// quadruple in array of
// integers

/* Function to find a maximum
   product of a quadruple
   in array of integers of size n */
function maxProduct($arr, $n)
{
    // if size is less than 4,
    // no quadruple exists
    if ($n < 4)
        return -1;

    // Sort the array
    // in ascending order
    sort($arr);

    $x = $arr[$n - 1] * $arr[$n - 2] *
         $arr[$n - 3] * $arr[$n - 4];
    $y = $arr[0] * $arr[1] *
         $arr[2] * $arr[3];
    $z = $arr[0] * $arr[1] *
```

```
        $arr[$n - 1] * $arr[$n - 2];

    // Return the maximum
    // of x, y and z
    return max($x, max($y, $z));
}

// Driver Code
$arr = array(-10, -3, 5, 6, -20);
$n = sizeof($arr);

$max = maxProduct($arr, $n);
    if ($max == -1)
        echo "No Quadruple Exists";
    else
        echo "Maximum product is " . $max;

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

Output :

```
Maximum product is 6000
```

Approach 3: O(n) Time, O(1) Space

1. Scan the array and compute Maximum, second maximum, third maximum and fourth maximum element present in the array.
2. Scan the array and compute Minimum, second minimum, third minimum and fourth minimum element present in the array.
3. Return the maximum of (product of Maximum, second maximum, third maximum and fourth maximum), (product of Minimum, second minimum, third minimum and fourth minimum) and (product of Maximum, second maximum, Minimum, second minimum).

Note – Step 1 and Step 2 can be done in single traversal of the array.

Below is its implementation–

C++

```
 // A O(n) C++ program to find maximum quadruple in
// an array.
#include <bits/stdc++.h>
using namespace std;

/* Function to find a maximum product of a quadruple
```

```
   in array of integers of size n */
int maxProduct(int arr[], int n)
{
    // if size is less than 4, no quadruple exists
    if (n < 4)
        return -1;

    // Initialize Maximum, second maximum, third
    // maximum and fourth maximum element
    int maxA = INT_MIN, maxB = INT_MIN,
        maxC = INT_MIN, maxD = INT_MIN;

    // Initialize Minimum, second minimum, third
    // minimum and fourth minimum element
    int minA = INT_MAX, minB = INT_MAX,
        minC = INT_MAX, minD = INT_MAX;

    for (int i = 0; i < n; i++) {

        // Update Maximum, second maximum, third
        // maximum and fourth maximum element
        if (arr[i] > maxA) {
            maxD = maxC;
            maxC = maxB;
            maxB = maxA;
            maxA = arr[i];
        }

        // Update second maximum, third maximum
        // and fourth maximum element
        else if (arr[i] > maxB) {
            maxD = maxC;
            maxC = maxB;
            maxB = arr[i];
        }

        // Update third maximum and
        // fourth maximum element
        else if (arr[i] > maxC) {
            maxD = maxC;
            maxC = arr[i];
        }

        // Update fourth maximum element
        else if (arr[i] > maxD)
            maxD = arr[i];

        // Update Minimum, second minimum
```

```
        // third minimum and fourth minimum element
        if (arr[i] < minA) {
            minD = minC;
            minC = minB;
            minB = minA;
            minA = arr[i];
        }

        // Update second minimum, third
        // minimum and fourth minimum element
        else if (arr[i] < minB) {
            minD = minC;
            minC = minB;
            minB = arr[i];
        }

        // Update third minimum and
        // fourth minimum element
        else if (arr[i] < minC) {
            minD = minC;
            minC = arr[i];
        }

        // Update fourth minimum element
        else if (arr[i] < minD)
            minD = arr[i];
    }

    int x = maxA * maxB * maxC * maxD;
    int y = minA * minB * minC * minD;
    int z = minA * minB * maxA * maxB;
    // Return the maximum of x, y and z
    return max(x, max(y, z));
}

// Driver program to test above function
int main()
{
    int arr[] = { 1, -4, 3, -6, 7, 0 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int max = maxProduct(arr, n);
    if (max == -1)
        cout << "No Quadruple Exists";
    else
        cout << "Maximum product is " << max;
    return 0;
}
```

Output :

```
Maximum product is 504
```

Improved By : vt_m, mithunkumarmnnit321, Abby_akku

Source

https://www.geeksforgeeks.org/maximum-product-quadruple-sub-sequence-size-4-array/

Chapter 163

Maximum sum of elements from each row in the matrix

Maximum sum of elements from each row in the matrix - GeeksforGeeks

Given a matrix, find the maximum sum we can have by selecting just one element from every row. Condition is element selected from nth row must be strictly greater than element from (n-1)th row, else no element must be taken from row. Print the sum if possible else print -1.

Examples :

```
Input :
1 2 3
1 2 3
7 8 9
Output : 14 (2 + 3 + 9) (values we
are adding are strictly increasing)

Input :
4 2 3
3 2 1
1 2 2
Output : -1
(No subsequent increasing elements
can be picked from consecutive rows)
```

Approach :- One can simply run the loop from last row, get the greatest element from there say it prev_max, and keep record for the minimum difference among the elements of the row just above it, if any element found with positive difference, then add it to prev_max else print -1. Continue the same process for every row.

C++

```
 // CPP Program to find row-wise maximum element
// sum considering elements in increasing oder.
#include <bits/stdc++.h>
#define N 3
using namespace std;

// Function to perform given task
int getGreatestSum(int a[][N])
{
    // Getting the maximum element from last row
    int prev_max = 0;
    for (int j = 0; j < N; j++)
        if (prev_max < a[N - 1][j])
            prev_max = a[N - 1][j];

    // Comparing it with the elements of above rows
    int sum = prev_max;
    for (int i = N - 2; i >= 0; i--) {

        // Maximum of current row.
        int curr_max = INT_MIN;
        for (int j = 0; j < N; j++)
            if (prev_max > a[i][j] && a[i][j] > curr_max)
                curr_max = a[i][j];

        // If we could not an element smaller
        // than prev_max.
        if (curr_max == INT_MAX)
            return -1;

        prev_max = curr_max;
        sum += prev_max;
    }
    return sum;
}

// Driver code
int main()
{
    int a[3][3] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
    cout << getGreatestSum(a) << endl;
    int b[3][3] = { { 4, 5, 6 }, { 4, 5, 6 }, { 4, 5, 6 } };
    cout << getGreatestSum(b) << endl;
    return 0;
}
```

Java

```
 // Java Program to find row-wise maximum
// element sum considering elements in
// increasing oder.
class GFG {

    static final int N = 3;

    // Function to perform given task
    static int getGreatestSum(int a[][]) {

        // Getting the maximum element from
        // last row
        int prev_max = 0;

        for (int j = 0; j < N; j++)
            if (prev_max < a[N - 1][j])
                prev_max = a[N - 1][j];

        // Comparing it with the elements
        // of above rows
        int sum = prev_max;

        for (int i = N - 2; i >= 0; i--) {

            // Maximum of current row.
            int curr_max = -2147483648;

            for (int j = 0; j < N; j++)
                if (prev_max > a[i][j] &&
                        a[i][j] > curr_max)
                    curr_max = a[i][j];

            // If we could not an element smaller
            // than prev_max.
            if (curr_max == 2147483647)
                return -1;

            prev_max = curr_max;
            sum += prev_max;
        }

        return sum;
    }

    // Driver Program to test above function
    public static void main(String arg[]) {

        int a[][] = {{1, 2, 3},
```

```
                    {4, 5, 6},
                    {7, 8, 9}};
        System.out.println(getGreatestSum(a));

        int b[][] = {{4, 5, 6},
                    {4, 5, 6},
                    {4, 5, 6}};
        System.out.println(getGreatestSum(b));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python Program to find
# row-wise maximum element
# sum considering elements
# in increasing oder.

N = 3

# Function to perform given task
def getGreatestSum(a):

    # Getting the maximum
    # element from last row
    prev_max = 0
    for j in range(N):
        if (prev_max < a[N - 1][j]):
            prev_max = a[N - 1][j]

    # Comparing it with the
    # elements of above rows
    sum = prev_max
    for i in range(N - 2,-1,-1):

        #Maximum of current row.
        curr_max = -2147483648
        for j in range(N):
            if (prev_max > a[i][j] and a[i][j] > curr_max):
                curr_max = a[i][j]

        # If we could not an element smaller
        # than prev_max.
        if (curr_max == +2147483647):
            return -1
```

```
        prev_max = curr_max
        sum =sum+ prev_max

    return sum

# Driver code

a= [ [ 1, 2, 3 ],
     [ 4, 5, 6 ],
     [ 7, 8, 9 ] ]

print(getGreatestSum(a))

b = [ [ 4, 5, 6 ],
      [ 4, 5, 6 ],
      [ 4, 5, 6 ] ]

print(getGreatestSum(b))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# Program to find row-wise maximum
// element sum considering elements in
// increasing oder.
using System;

class GFG {

    static int N = 3;

    // Function to perform given task
    static int getGreatestSum(int [,]a) {

        // Getting the maximum element from
        // last row
        int prev_max = 0;

        for (int j = 0; j < N; j++)
            if (prev_max < a[N - 1, j])
                prev_max = a[N - 1, j];

        // Comparing it with the elements
        // of above rows
        int sum = prev_max;
```

```
        for (int i = N - 2; i >= 0; i--) {

            // Maximum of current row.
            int curr_max = -2147483648;

            for (int j = 0; j < N; j++)
                if (prev_max > a[i, j] &&
                        a[i, j] > curr_max)
                    curr_max = a[i, j];

            // If we could not an element smaller
            // than prev_max.
            if (curr_max == 2147483647)
                return -1;

            prev_max = curr_max;
            sum += prev_max;
        }

        return sum;
    }

    // Driver Program
    public static void Main() {

        int [,]a = {{1, 2, 3},
                    {4, 5, 6},
                    {7, 8, 9}};
        Console.WriteLine(getGreatestSum(a));

        int [,]b = {{4, 5, 6},
                    {4, 5, 6},
                    {4, 5, 6}};
        Console.WriteLine(getGreatestSum(b));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find
// row-wise maximum element
// sum considering elements
// in increasing order.

$N = 3;
```

```
// Function to perform given task
function getGreatestSum( $a)
{
    global $N;

    // Getting the maximum
    // element from last row
    $prev_max = 0;

    for ($j = 0; $j < $N; $j++)
        if ($prev_max < $a[$N - 1][$j])
            $prev_max = $a[$N - 1][$j];

    // Comparing it with the
    // elements of above rows
    $sum = $prev_max;
    for ($i = $N - 2; $i >= 0; $i--)
    {

        // Maximum of current row.
        $curr_max = PHP_INT_MIN;
        for ( $j = 0; $j < $N; $j++)
            if ($prev_max > $a[$i][$j] and
                $a[$i][$j] > $curr_max)
                $curr_max = $a[$i][$j];

        // If we could not an element
        // smaller than prev_max.
        if ($curr_max == PHP_INT_MAX)
            return -1;

        $prev_max = $curr_max;
        $sum += $prev_max;
    }
    return $sum;
}

// Driver code
$a = array(array(1, 2, 3),
           array(4, 5, 6),
           array(7, 8, 9));

echo getGreatestSum($a) ,"\n";
$b = array(array(4, 5, 6),
           array(4, 5, 6),
           array(4, 5, 6));
```

```
echo getGreatestSum($b) ,"\n";

// This code is contributed by anuj_67.
?>
```

Output :

```
18
15
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/maximum-sum-elements-row-matrix/

Chapter 164

Maximum sum of increasing order elements from n arrays

Maximum sum of increasing order elements from n arrays - GeeksforGeeks

Given n arrays of size m eacg. Find maximum sum obtained by selecting a number from each array such that the elements selected from i-th array is more than the element selected from (i-1)-th array. If maximum sum can not be obtained then return 0.

Examples:

```
Input : arr[][] = {{1, 7, 3, 4},
                   {4, 2, 5, 1},
                   {9, 5, 1, 8}}
Output : 18
Explanation :
We can select 4 from first array, 5 from
second array and 9 from third array.

Input : arr[][] = {{9, 8, 7},
                   {6, 5, 4},
                   {3, 2, 1}}
Output : 0
```

The idea is to start picking from last array. We pick the maximum element from last array, then we move to second last array. In second last array, we find the largest element which is smaller than the maximum element picked from last array. We repeat this process till we reach first array.

To obtain maximum sum we can sort all arrays and start bottom to up traversing each array from right to left and choose a number such that it is greater then previous element. If we are not able to select an element from array then return 0.

C++

```
 // CPP program to find maximum sum
// by selecting a element from n arrays
#include <bits/stdc++.h>
#define M 4
using namespace std;

// To calculate maximum sum by
// selecting element from each array
int maximumSum(int a[][M], int n) {

  // Sort each array
  for (int i = 0; i < n; i++)
    sort(a[i], a[i] + M);

  // Store maximum element
  // of last array
  int sum = a[n - 1][M - 1];
  int prev = a[n - 1][M - 1];
  int i, j;

  // Selecting maximum element from
  // previoulsy selected element
  for (i = n - 2; i >= 0; i--) {
    for (j = M - 1; j >= 0; j--) {
      if (a[i][j] < prev) {
        prev = a[i][j];
        sum += prev;
        break;
      }
    }

    // j = -1 means no element is
    // found in a[i] so return 0
    if (j == -1)
      return 0;
  }

  return sum;
}

// Driver program to test maximumSum
int main() {
  int arr[][M] = {{1, 7, 3, 4},
                  {4, 2, 5, 1},
                  {9, 5, 1, 8}};
  int n = sizeof(arr) / sizeof(arr[0]);
```

```
  cout << maximumSum(arr, n);
  return 0;
}
```

Java

```
 // Java program to find
// maximum sum by selecting
// a element from n arrays
import java.io.*;

class GFG
{
    static int M = 4;
    static int arr[][] = {{1, 7, 3, 4},
                          {4, 2, 5, 1},
                          {9, 5, 1, 8}};

    static void sort(int a[][],
                     int row, int n)
    {
        for (int i = 0; i < M - 1; i++)
        {
            if(a[row][i] > a[row][i + 1])
            {
                int temp = a[row][i];
                a[row][i] = a[row][i + 1];
                a[row][i + 1] = temp;
            }
        }
    }

    // To calculate maximum
    // sum by selecting element
    // from each array
    static int maximumSum(int a[][],
                          int n)
    {

    // Sort each array
    for (int i = 0; i < n; i++)
        sort(a, i, n);

    // Store maximum element
    // of last array
    int sum = a[n - 1][M - 1];
    int prev = a[n - 1][M - 1];
    int i, j;
```

```
    // Selecting maximum element
    // from previoulsy selected
    // element
    for (i = n - 2; i >= 0; i--)
    {
        for (j = M - 1; j >= 0; j--)
        {
            if (a[i][j] < prev)
            {
                prev = a[i][j];
                sum += prev;
                break;
            }
        }

        // j = -1 means no element
        // is found in a[i] so
        // return 0
        if (j == -1)
        return 0;
    }
    return sum;
    }

    // Driver Code
    public static void main(String args[])
    {
        int n = arr.length;
        System.out.print(maximumSum(arr, n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Python3

```
 # Python3 program to find
# maximum sum by selecting
# a element from n arrays
M = 4;

# To calculate maximum sum
# by selecting element from
# each array
def maximumSum(a, n) :
```

```
    global M;

    # Sort each array
    for i in range(0, n) :
        a[i].sort();

    # Store maximum element
    # of last array
    sum = a[n - 1][M - 1];
    prev = a[n - 1][M - 1];

    # Selecting maximum
    # element from previoulsy
    # selected element
    for i in range(n - 2,
                   -1, -1) :

        for j in range(M - 1,
                       -1, -1) :

            if (a[i][j] < prev) :

                prev = a[i][j];
                sum += prev;
                break;

        # j = -1 means no element
        # is found in a[i] so
        # return 0
        if (j == -1) :
            return 0;
    return sum;

# Driver Code
arr = [[1, 7, 3, 4],
       [4, 2, 5, 1],
       [9, 5, 1, 8]];
n = len(arr) ;
print (maximumSum(arr, n));

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# program to find maximum
// sum by selecting a element
// from n arrays
```

```
using System;

class GFG
{
    static int M = 4;

    static void sort(ref int[,] a,
                     int row, int n)
    {
        for (int i = 0; i < M-1; i++)
        {
            if(a[row, i] > a[row, i + 1])
            {
                int temp = a[row, i];
                a[row, i] = a[row, i + 1];
                a[row, i + 1] = temp;
            }
        }
    }

    // To calculate maximum
    // sum by selecting
    // element from each array
    static int maximumSum(int[,] a,
                          int n)
    {

    int i = 0, j = 0;

    // Sort each array
    for (i = 0; i < n; i++)
        sort(ref a, i, n);

    // Store maximum element
    // of last array
    int sum = a[n - 1, M - 1];
    int prev = a[n - 1, M - 1];


    // Selecting maximum element
    // from previoulsy selected
    // element
    for (i = n - 2; i >= 0; i--)
    {
        for (j = M - 1; j >= 0; j--)
        {
        if (a[i, j] < prev)
        {
```

```
            prev = a[i, j];
            sum += prev;
            break;
        }
        }

        // j = -1 means no element
        // is found in a[i] so
        // return 0
        if (j == -1)
        return 0;
    }

    return sum;
    }

    // Driver Code
    static void Main()
    {
    int [,]arr = new int[,]{{1, 7, 3, 4},
                            {4, 2, 5, 1},
                            {9, 5, 1, 8}};
    int n = arr.GetLength(0);
    Console.Write(maximumSum(arr, n));
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

PHP

```
 <?php
// PHP program to find maximum
// sum by selecting a element
// from n arrays
$M = 4;

// To calculate maximum sum
// by selecting element from
// each array
function maximumSum($a, $n)
{
    global $M;

    // Sort each array
    for ($i = 0; $i < $n; $i++)
        sort($a[$i]);
```

```
    // Store maximum element
    // of last array
    $sum = $a[$n - 1][$M - 1];
    $prev = $a[$n - 1][$M - 1];
    $i; $j;

    // Selecting maximum element from
    // previoulsy selected element
    for ($i = $n - 2; $i >= 0; $i--)
    {
        for ($j = $M - 1; $j >= 0; $j--)
        {
        if ($a[$i][$j] < $prev)
        {
            $prev = $a[$i][$j];
            $sum += $prev;
            break;
        }
        }

        // j = -1 means no element is
        // found in a[i] so return 0
        if ($j == -1)
        return 0;
    }

    return $sum;
}

// Driver Code
$arr = array(array(1, 7, 3, 4),
             array(4, 2, 5, 1),
             array(9, 5, 1, 8));
$n = sizeof($arr) ;
echo maximumSum($arr, $n);

// This code is contributed by m_kit
?>
```

Output:

```
18
```

Worst Case Time Complexity : O(mn Log m)

We can optimize above solution to work in O(mn). We can skip sorting to find the maximum elements.

C++

```
 // CPP program to find maximum sum
// by selecting a element from n arrays
#include <bits/stdc++.h>
#define M 4
using namespace std;

// To calculate maximum sum by
// selecting element from each array
int maximumSum(int a[][M], int n) {

  // Store maximum element of last array
  int prev = *max_element(&a[n-1][0],
                  &a[n-1][M-1] + 1);

  // Selecting maximum element from
  // previoulsy selected element
  int sum = prev;
  for (int i = n - 2; i >= 0; i--) {

    int max_smaller = INT_MIN;
    for (int j = M - 1; j >= 0; j--) {
      if (a[i][j] < prev &&
          a[i][j] > max_smaller)
        max_smaller = a[i][j];
    }

    // max_smaller equals to INT_MIN means
    // no element is found in a[i] so
    // return 0
    if (max_smaller == INT_MIN)
      return 0;

    prev = max_smaller;
    sum += max_smaller;
  }

  return sum;
}

// Driver program to test maximumSum
int main() {
  int arr[][M] = {{1, 7, 3, 4},
                  {4, 2, 5, 1},
                  {9, 5, 1, 8}};
  int n = sizeof(arr) / sizeof(arr[0]);
  cout << maximumSum(arr, n);
```

```
  return 0;
}
```

Output:

```
18
```

Time Complexity: O(mn)

Improved By : jit_t, manishshaw1

Source

https://www.geeksforgeeks.org/maximum-sum-increasing-order-elements-n-arrays/

Chapter 165

Median of two sorted arrays of different sizes

Median of two sorted arrays of different sizes - GeeksforGeeks

This is an extension of median of two sorted arrays of equal size problem. Here we handle arrays of unequal size also.

The approach discussed in this post is similar to method 2 of equal size post. The basic idea is same, we find the median of two arrays and compare the medians to discard almost half of the elements in both arrays. Since the number of elements may differ here, there are many base cases that need to be handled separately. Before we proceed to complete solution, let us first talk about all base cases.

Let the two arrays be A[N] and B[M]. In the following explanation, it is assumed that N is smaller than or equal to M.

Base cases:
The smaller array has only one element
Case 0: N = 0, M = 2
Case 1: N = 1, M = 1.
Case 2: N = 1, M is odd
Case 3: N = 1, M is even
The smaller array has only two elements
Case 4: N = 2, M = 2
Case 5: N = 2, M is odd
Case 6: N = 2, M is even

Case 0: There are no elements in first array, return median of second array. If second array is also empty, return -1.

Case 1: There is only one element in both arrays, so output the average of A[0] and B[0].

Case 2: N = 1, M is odd
Let B[5] = {5, 10, 12, 15, 20}
First find the middle element of B[], which is 12 for above array. There are following 4

sub-cases.
…**2.1** If A[0] is smaller than 10, the median is average of 10 and 12.
…**2.2** If A[0] lies between 10 and 12, the median is average of A[0] and 12.
…**2.3** If A[0] lies between 12 and 15, the median is average of 12 and A[0].
…**2.4** If A[0] is greater than 15, the median is average of 12 and 15.
In all the sub-cases, we find that 12 is fixed. So, we need to find the median of B[M / 2 – 1], B[M / 2 + 1], A[0] and take its average with B[M / 2].

Case 3: N = 1, M is even
Let B[4] = {5, 10, 12, 15}
First find the middle items in B[], which are 10 and 12 in above example. There are following 3 sub-cases.
…**3.1** If A[0] is smaller than 10, the median is 10.
…**3.2** If A[0] lies between 10 and 12, the median is A[0].
…**3.3** If A[0] is greater than 12, the median is 12.
So, in this case, find the median of three elements B[M / 2 – 1], B[M / 2] and A[0].

Case 4: N = 2, M = 2
There are four elements in total. So we find the median of 4 elements.

Case 5: N = 2, M is odd
Let B[5] = {5, 10, 12, 15, 20}
The median is given by median of following three elements: B[M/2], max(A[0], B[M/2 – 1]), min(A[1], B[M/2 + 1]).

Case 6: N = 2, M is even
Let B[4] = {5, 10, 12, 15}
The median is given by median of following four elements: B[M/2], B[M/2 – 1], max(A[0], B[M/2 – 2]), min(A[1], B[M/2 + 1])

Remaining Cases:
Once we have handled the above base cases, following is the remaining process.
1) Find the middle item of A[] and middle item of B[].
…..**1.1)** If the middle item of A[] is greater than middle item of B[], ignore the last half of A[], let length of ignored part is idx. Also, cut down B[] by idx from the start.
…..**1.2)** else, ignore the first half of A[], let length of ignored part is idx. Also, cut down B[] by idx from the last.

Following is implementation of the above approach.

C++

```
 // A C++ program to find median of two sorted arrays of
// unequal sizes
#include <bits/stdc++.h>
using namespace std;

// A utility function to find median of two integers
float MO2(int a, int b)
{ return ( a + b ) / 2.0; }
```

```
// A utility function to find median of three integers
float MO3(int a, int b, int c)
{
    return a + b + c - max(a, max(b, c))
                     - min(a, min(b, c));
}

// A utility function to find median of four integers
float MO4(int a, int b, int c, int d)
{
    int Max = max( a, max( b, max( c, d ) ) );
    int Min = min( a, min( b, min( c, d ) ) );
    return ( a + b + c + d - Max - Min ) / 2.0;
}

// Utility function to find median of single array
float medianSingle(int arr[], int n)
{
   if (n == 0)
      return -1;
   if (n%2 == 0)
        return (double)(arr[n/2] + arr[n/2-1])/2;
   return arr[n/2];
}

// This function assumes that N is smaller than or equal to M
// This function returns -1 if both arrays are empty
float findMedianUtil( int A[], int N, int B[], int M )
{
    // If smaller array is empty, return median from second array
    if (N == 0)
      return medianSingle(B, M);

    // If the smaller array has only one element
    if (N == 1)
    {
        // Case 1: If the larger array also has one element,
        // simply call MO2()
        if (M == 1)
            return MO2(A[0], B[0]);

        // Case 2: If the larger array has odd number of elements,
        // then consider the middle 3 elements of larger array and
        // the only element of smaller array. Take few examples
        // like following
        // A = {9}, B[] = {5, 8, 10, 20, 30} and
        // A[] = {1}, B[] = {5, 8, 10, 20, 30}
```

```
    if (M & 1)
        return MO2( B[M/2], MO3(A[0], B[M/2 - 1], B[M/2 + 1]) );

    // Case 3: If the larger array has even number of element,
    // then median will be one of the following 3 elements
    // ... The middle two elements of larger array
    // ... The only element of smaller array
    return MO3( B[M/2], B[M/2 - 1], A[0] );
}

// If the smaller array has two elements
else if (N == 2)
{
    // Case 4: If the larger array also has two elements,
    // simply call MO4()
    if (M == 2)
        return MO4(A[0], A[1], B[0], B[1]);

    // Case 5: If the larger array has odd number of elements,
    // then median will be one of the following 3 elements
    // 1. Middle element of larger array
    // 2. Max of first element of smaller array and element
    //    just before the middle in bigger array
    // 3. Min of second element of smaller array and element
    //    just after the middle in bigger array
    if (M & 1)
        return MO3 ( B[M/2],
                     max(A[0], B[M/2 - 1]),
                     min(A[1], B[M/2 + 1])
                   );

    // Case 6: If the larger array has even number of elements,
    // then median will be one of the following 4 elements
    // 1) & 2) The middle two elements of larger array
    // 3) Max of first element of smaller array and element
    //    just before the first middle element in bigger array
    // 4. Min of second element of smaller array and element
    //    just after the second middle in bigger array
    return MO4 ( B[M/2],
                 B[M/2 - 1],
                 max( A[0], B[M/2 - 2] ),
                 min( A[1], B[M/2 + 1] )
               );
}

int idxA = ( N - 1 ) / 2;
int idxB = ( M - 1 ) / 2;
```

```
     /* if A[idxA] <= B[idxB], then median must exist in
        A[idxA....] and B[....idxB] */
    if (A[idxA] <= B[idxB] )
      return findMedianUtil(A + idxA, N/2 + 1, B, M - idxA );

    /* if A[idxA] > B[idxB], then median must exist in
       A[...idxA] and B[idxB....] */
    return findMedianUtil(A, N/2 + 1, B + idxA, M - idxA );
}

// A wrapper function around findMedianUtil(). This function
// makes sure that smaller array is passed as first argument
// to findMedianUtil
float findMedian( int A[], int N, int B[], int M )
{
    if (N > M)
       return findMedianUtil( B, M, A, N );

    return findMedianUtil( A, N, B, M );
}

// Driver program to test above functions
int main()
{
    int A[] = {900};
    int B[] = {5, 8, 10, 20};

    int N = sizeof(A) / sizeof(A[0]);
    int M = sizeof(B) / sizeof(B[0]);

    printf("%f", findMedian( A, N, B, M ) );
    return 0;
}
```

PHP

```
 <?php
// A PHP program to find median
// of two sorted arrays of
// unequal sizes

// A utility function to
// find median of two integers
function MO2($a, $b)
{
    return ($a + $b) / 2.0;
}
```

```
// A utility function to
// find median of three integers
function MO3($a, $b, $c)
{
    return $a + $b + $c -
       max($a, max($b, $c)) -
       min($a, min($b, $c));
}

// A utility function to find
// median of four integers
function MO4($a, $b, $c, $d)
{
    $Max = max($a, max($b, max($c, $d)));
    $Min = min($a, min($b, min( $c, $d)));
    return ($a + $b + $c + $d - $Max - $Min) / 2.0;
}

// Utility function to
// find median of single array
function medianSingle($arr, $n)
{
if ($n == 0)
    return -1;
if ($n % 2 == 0)
        return ($arr[$n / 2] +
                $arr[$n / 2 - 1]) / 2;
return $arr[$n / 2];
}

// This function assumes that N
// is smaller than or equal to M
// This function returns -1 if
// both arrays are empty
function findMedianUtil(&$A, $N, &$B, $M )
{
    // If smaller array is empty,
    // return median from second array
    if ($N == 0)
    return medianSingle($B, $M);

    // If the smaller array
    // has only one element
    if ($N == 1)
    {
        // Case 1: If the larger
        // array also has one
        // element, simply call MO2()
```

```
    if ($M == 1)
        return MO2($A[0], $B[0]);

    // Case 2: If the larger array
    // has odd number of elements,
    // then consider the middle 3
    // elements of larger array and
    // the only element of smaller
    // array. Take few examples
    // like following
    // $A = array(9),
    // $B = array(5, 8, 10, 20, 30)
    // and $A = array(1),
    // $B = array(5, 8, 10, 20, 30)
    if ($M & 1)
        return MO2($B[$M / 2], $MO3($A[0],
                   $B[$M / 2 - 1],
                   $B[$M / 2 + 1]));

    // Case 3: If the larger array
    // has even number of element,
    // then median will be one of
    // the following 3 elements
    // ... The middle two elements
    //     of larger array
    // ... The only element of
    //     smaller array
    return MO3($B[$M / 2],
               $B[$M / 2 - 1], $A[0]);
}

// If the smaller array
// has two elements
else if ($N == 2)
{
    // Case 4: If the larger
    // array also has two elements,
    // simply call MO4()
    if ($M == 2)
        return MO4($A[0], $A[1],
                   $B[0], $B[1]);

    // Case 5: If the larger array
    // has odd number of elements,
    // then median will be one of
    // the following 3 elements
    // 1. Middle element of
    //    larger array
```

```
    // 2. Max of first element of
    //    smaller array and element
    // just before the middle
    // in bigger array
    // 3. Min of second element
    //    of smaller array and element
    // just after the middle
    // in bigger array
    if ($M & 1)
        return MO3 ($B[$M / 2],
                    max($A[0], $B[$M / 2 - 1]),
                    min($A[1], $B[$M / 2 + 1]));

    // Case 6: If the larger array
    // has even number of elements,
    // then median will be one of
    // the following 4 elements
    // 1) & 2) The middle two
    // elements of larger array
    // 3) Max of first element of
    // smaller array and element
    // just before the first middle
    // element in bigger array
    // 4. Min of second element of
    // smaller array and element
    // just after the second
    // middle in bigger array
    return MO4 ($B[$M / 2],
                $B[$M / 2 - 1],
                max($A[0], $B[$M / 2 - 2]),
                min($A[1], $B[$M / 2 + 1]));
}

$idxA = ($N - 1 ) / 2;
$idxB = ($M - 1 ) / 2;

/* if $A[$idxA] <= $B[$idxB], then
    median must exist in
    $A[$idxA....] and $B[....$idxB] */
if ($A[$idxA] <= $B[$idxB] )
return findMedianUtil($A + $idxA,
                      $N / 2 + 1, $B,
                      $M - $idxA );

/* if $A[$idxA] > $B[$idxB],
then median must exist in
$A[...$idxA] and $B[$idxB....] */
return findMedianUtil($A, $N/2 + 1,
```

```
                        $B + $idxA, $M - $idxA );
}

// A wrapper function around
// findMedianUtil(). This
// function makes sure that
// smaller array is passed as
// first argument to findMedianUtil
function findMedian(&$A, $N,
                    &$B, $M )
{
    if ($N > $M)
    return findMedianUtil($B, $M,
                        $A, $N );

    return findMedianUtil($A, $N,
                        $B, $M );
}

// Driver Code
$A = array(900);
$B = array(5, 8, 10, 20);

$N = sizeof($A);
$M = sizeof($B);

echo findMedian( $A, $N, $B, $M );

// This code is contributed
// by ChitraNayal
?>
```

Output:

```
 10
```

Time Complexity: O(LogM + LogN)

Improved By : ChitraNayal

Source

https://www.geeksforgeeks.org/median-of-two-sorted-arrays-of-different-sizes/

Chapter 166

Median of two sorted arrays of different sizes | Set 1 (Linear)

Median of two sorted arrays of different sizes | Set 1 (Linear) - GeeksforGeeks

This is an extension of the median of two sorted arrays of equal size problem. Here we handle arrays of unequal size also.

Examples:

```
Input : a[] ={1, 12, 15, 26, 38}
        b[] = {2, 13, 24}
Output : 14
Explanation :
         After merging arrays the result is
         1, 2, 12, 13, 15, 24, 26, 38
         median = (13+15)/2 = 14.
```

The approach discussed in this post is similar to method 1 of equal size median. Use merge procedure of merge sort. Keep track of count while comparing elements of two arrays. If count becomes ()(n1+n2)/2 (For n1+n2 elements), we have reached the median. Take the average of the elements at indexes ((n1+n2)/2)-1 and (n1+n2)/2 in the merged array. See the below implementation.

C++

```
 // A C++ program to find median of two sorted arrays of
// unequal sizes
#include <bits/stdc++.h>
using namespace std;

// A utility function to find median of two integers
```

```
/* This function returns median of a[] and b[].
Assumptions in this function: Both a[] and b[]
are sorted arrays  */
float findmedian(int a[], int n1, int b[], int n2)
{
    int i = 0; /* Current index of
             i/p array a[] */
    int j = 0; /* Current index of
                   i/p array b[] */
    int k;
    int m1 = -1, m2 = -1;
    for (k = 0; k <= (n1 + n2) / 2; k++) {

        if (i < n1 && j < n2) {
            if (a[i] < b[j]) {
                m2 = m1;
                m1 = a[i];
                i++;
            }
            else {
                m2 = m1;
                m1 = b[j];
                j++;
            }
        }

        /* Below is to handle the case where
           all elements of a[] are
           smaller than smallest(or first)
           element of b[] or a[] is empty*/
        else if (i == n1) {
            m2 = m1;
            m1 = b[j];
            j++;
        }

        /* Below is to handle case where
           all elements of b[] are
           smaller than smallest(or first)
           element of a[] or b[] is empty*/
        else if (j == n2) {
            m2 = m1;
            m1 = a[i];
            i++;
        }
    }

    /*Below is to handle the case where
```

```
     sum of number of elements
     of the arrays is even */
    if ((n1 + n2) % 2 == 0)
        return (m1 + m2) * 1.0 / 2;

    /* Below is to handle the case where
       sum of number of elements
       of the arrays is odd */
    return m1;
}

// Driver program to test above functions
int main()
{
    int a[] = { 1, 12, 15, 26, 38 };
    int b[] = { 2, 13, 24 };

    int n1 = sizeof(a) / sizeof(a[0]);
    int n2 = sizeof(b) / sizeof(b[0]);

    printf("%f", findmedian(a, n1, b, n2));

    return 0;
}
```

Java

```
 // A Java program to find median of two sorted arrays of
// unequal sizes

import java.io.*;

class GFG
{



// A utility function to find median of two integers
/* This function returns median of a[] and b[].
Assumptions in this function: Both a[] and b[]
are sorted arrays */
static float findmedian(int a[], int n1, int b[], int n2)
{
    int i = 0; /* Current index of
            i/p array a[] */
    int j = 0; /* Current index of
                i/p array b[] */
    int k;
```

```
int m1 = -1, m2 = -1;
for (k = 0; k <= (n1 + n2) / 2; k++)
{

    if (i < n1 && j < n2)
    {
        if (a[i] < b[j])
        {
            m2 = m1;
            m1 = a[i];
            i++;
        }
        else
        {
            m2 = m1;
            m1 = b[j];
            j++;
        }
    }

    /* Below is to handle the case where
    all elements of a[] are
    smaller than smallest(or first)
    element of b[] or a[] is empty*/
    else if (i == n1)
    {
        m2 = m1;
        m1 = b[j];
        j++;
    }

    /* Below is to handle case where
    all elements of b[] are
    smaller than smallest(or first)
    element of a[] or b[] is empty*/
    else if (j == n2)
    {
        m2 = m1;
        m1 = a[i];
        i++;
    }
}

/*Below is to handle the case where
sum of number of elements
of the arrays is even */
if ((n1 + n2) % 2 == 0)
{
```

```
        return (m1 + m2) *(float) 1.0 / 2;
    }
    /* Below is to handle the case where
    sum of number of elements
    of the arrays is odd */
    return m1;
}

// Driver program to test above functions
    public static void main (String[] args)
    {
       int a[] = {1, 12, 15, 26, 38 };
       int b[] = {2, 13, 24};

       int n1 = a.length;
       int n2 = b.length;

       System.out.println( findmedian(a, n1, b, n2));
    }
}

// This code has been contributed by inder_verma.
```

C#

```
 // A C# program to find median
// of two sorted arrays of
// unequal sizes
using System;

class GFG
{

// A utility function to find
// median of two integers
/* This function returns
   median of a[] and b[].
   Assumptions in this
   function: Both a[] and b[]
   are sorted arrays */
static float findmedian(int []a, int n1,
                        int []b, int n2)
{
int i = 0; /* Current index of
              i/p array a[] */
int j = 0; /* Current index of
              i/p array b[] */
int k;
```

```
int m1 = -1, m2 = -1;
for (k = 0; k <= (n1 + n2) / 2; k++)
{
    if (i < n1 && j < n2)
    {
        if (a[i] < b[j])
        {
            m2 = m1;
            m1 = a[i];
            i++;
        }
        else
        {
            m2 = m1;
            m1 = b[j];
            j++;
        }
    }

    /* Below is to handle the case where
    all elements of a[] are
    smaller than smallest(or first)
    element of b[] or a[] is empty*/
    else if (i == n1)
    {
        m2 = m1;
        m1 = b[j];
        j++;
    }

    /* Below is to handle case where
    all elements of b[] are
    smaller than smallest(or first)
    element of a[] or b[] is empty*/
    else if (j == n2)
    {
        m2 = m1;
        m1 = a[i];
        i++;
    }
}

/*Below is to handle the case where
sum of number of elements
of the arrays is even */
if ((n1 + n2) % 2 == 0)
{
    return (m1 + m2) * (float) 1.0 / 2;
```

```
}
/* Below is to handle the case
where sum of number of elements
of the arrays is odd */
return m1;
}

// Driver Code
public static void Main ()
{
    int []a = {1, 12, 15, 26, 38 };
    int []b = {2, 13, 24};

    int n1 = a.Length;
    int n2 = b.Length;

    Console.WriteLine(findmedian(a, n1, b, n2));
}
}

// This code is contributed
// by Subhadeep Gupta
```

PHP

```
 <?php
// A PHP program to find median of
// two sorted arrays of unequal sizes

// A utility function to find
// median of two integers
/* This function returns median
of a[] and b[]. Assumptions in this
function: Both a[] and b[] are
sorted arrays */
function findmedian($a, $n1, $b, $n2)
{
    $i = 0; /* Current index of
               i/p array a[] */
    $j = 0; /* Current index of
               i/p array b[] */
    $k;
    $m1 = -1; $m2 = -1;
    for ($k = 0;
         $k <= ($n1 + $n2) / 2; $k++)
    {

        if ($i < $n1 and $j < $n2)
```

```
    {
        if ($a[$i] < $b[$j])
        {
            $m2 = $m1;
            $m1 = $a[$i];
            $i++;
        }
        else
        {
            $m2 = $m1;
            $m1 = $b[$j];
            $j++;
        }
    }

    /* Below is to handle the case
    where all elements of a[] are
    smaller than smallest(or first)
    element of b[] or a[] is empty*/
    else if (i == n1)
    {
        $m2 = $m1;
        $m1 = $b[j];
        $j++;
    }

    /* Below is to handle case
    where all elements of b[] are
    smaller than smallest(or first)
    element of a[] or b[] is empty*/
    else if ($j == $n2)
    {
        $m2 = $m1;
        $m1 = $a[$i];
        $i++;
    }
}

/*Below is to handle the case
where sum of number of elements
of the arrays is even */
if (($n1 + $n2) % 2 == 0)
    return ($m1 + $m2) * 1.0 / 2;

/* Below is to handle the case
where sum of number of elements
of the arrays is odd */
return m1;
```

```
}

// Driver Code
$a = array( 1, 12, 15, 26, 38 );
$b = array( 2, 13, 24 );

$n1 = count($a);
$n2 = count($b);

echo(findmedian($a, $n1, $b, $n2));

// This code is contributed
// by inder_verma.
?>
```

Output:

```
14.000000
```

Time Complexity: O(n)

More Efficient (Log time complexity methods)

1. Median of two sorted arrays of different sizes.
2. Median of two sorted arrays of different sizes in min(Log (n1, n2))

Improved By : inderDuMCA, tufan_gupta2000

Source

https://www.geeksforgeeks.org/median-of-two-sorted-arrays-of-different-sizes-set-1-linear/

Chapter 167

Median of two sorted arrays of same size

Median of two sorted arrays of same size - GeeksforGeeks

There are 2 sorted arrays A and B of size n each. Write an algorithm to find the median of the array obtained after merging the above 2 arrays(i.e. array of length 2n). The complexity should be O(log(n)).

```
Input : ar1[] = {1, 12, 15, 26, 38}
        ar2[] = {2, 13, 17, 30, 45}
Output : 16

Explanation :
After merging two arrays, we get
{1, 2, 12, 13, 15. 17, 26, 30, 38, 45}
Middle two elements are 15 and 17
Average of middle elements is (15 + 17)/2
which is equal to 16
```

Note : Since size of the set for which we are looking for median is even (2n), we need take average of middle two numbers and return floor of the average.

Method 1 (Simply count while Merging)
Use merge procedure of merge sort. Keep track of count while comparing elements of two arrays. If count becomes n(For 2n elements), we have reached the median. Take the average of the elements at indexes n-1 and n in the merged array. See the below implementation.

C++

```
// A Simple Merge based O(n)
```

```
// solution to find median of
// two sorted arrays
#include <bits/stdc++.h>
using namespace std;

/* This function returns
median of ar1[] and ar2[].
Assumptions in this function:
Both ar1[] and ar2[]
are sorted arrays
Both have n elements */
int getMedian(int ar1[],
              int ar2[], int n)
{
    int i = 0; /* Current index of
                  i/p array ar1[] */
    int j = 0; /* Current index of
                  i/p array ar2[] */
    int count;
    int m1 = -1, m2 = -1;

    /* Since there are 2n elements,
    median will be average of elements
    at index n-1 and n in the array
    obtained after merging ar1 and ar2 */
    for (count = 0; count <= n; count++)
    {
        /* Below is to handle case where
           all elements of ar1[] are
           smaller than smallest(or first)
           element of ar2[]*/
        if (i == n)
        {
            m1 = m2;
            m2 = ar2[0];
            break;
        }

        /*Below is to handle case where
          all elements of ar2[] are
          smaller than smallest(or first)
          element of ar1[]*/
        else if (j == n)
        {
            m1 = m2;
            m2 = ar1[0];
            break;
        }
```

```
        if (ar1[i] < ar2[j])
        {
            /* Store the prev median */
            m1 = m2;
            m2 = ar1[i];
            i++;
        }
        else
        {
            /* Store the prev median */
            m1 = m2;
            m2 = ar2[j];
            j++;
        }
    }

    return (m1 + m2)/2;
}

// Driver Code
int main()
{
    int ar1[] = {1, 12, 15, 26, 38};
    int ar2[] = {2, 13, 17, 30, 45};

    int n1 = sizeof(ar1) / sizeof(ar1[0]);
    int n2 = sizeof(ar2) / sizeof(ar2[0]);
    if (n1 == n2)
        cout << "Median is "
             << getMedian(ar1, ar2, n1) ;
    else
        cout << "Doesn't work for arrays"
             << " of unequal size" ;
    getchar();
    return 0;
}

// This code is contributed
// by Shivi_Aggarwal
```

C

```
 // A Simple Merge based O(n) solution to find median of
// two sorted arrays
#include <stdio.h>

/* This function returns median of ar1[] and ar2[].
```

```
   Assumptions in this function:
   Both ar1[] and ar2[] are sorted arrays
   Both have n elements */
int getMedian(int ar1[], int ar2[], int n)
{
    int i = 0;  /* Current index of i/p array ar1[] */
    int j = 0; /* Current index of i/p array ar2[] */
    int count;
    int m1 = -1, m2 = -1;

    /* Since there are 2n elements, median will be average
     of elements at index n-1 and n in the array obtained after
     merging ar1 and ar2 */
    for (count = 0; count <= n; count++)
    {
        /*Below is to handle case where all elements of ar1[] are
          smaller than smallest(or first) element of ar2[]*/
        if (i == n)
        {
            m1 = m2;
            m2 = ar2[0];
            break;
        }

        /*Below is to handle case where all elements of ar2[] are
          smaller than smallest(or first) element of ar1[]*/
        else if (j == n)
        {
            m1 = m2;
            m2 = ar1[0];
            break;
        }

        if (ar1[i] < ar2[j])
        {
            m1 = m2;  /* Store the prev median */
            m2 = ar1[i];
            i++;
        }
        else
        {
            m1 = m2;  /* Store the prev median */
            m2 = ar2[j];
            j++;
        }
    }

    return (m1 + m2)/2;
```

```
}

/* Driver program to test above function */
int main()
{
    int ar1[] = {1, 12, 15, 26, 38};
    int ar2[] = {2, 13, 17, 30, 45};

    int n1 = sizeof(ar1)/sizeof(ar1[0]);
    int n2 = sizeof(ar2)/sizeof(ar2[0]);
    if (n1 == n2)
        printf("Median is %d", getMedian(ar1, ar2, n1));
    else
        printf("Doesn't work for arrays of unequal size");
    getchar();
    return 0;
}
```

Java

```
 // A Simple Merge based O(n) solution
// to find median of two sorted arrays

class Main
{
    // function to calculate median
    static int getMedian(int ar1[], int ar2[], int n)
    {
        int i = 0;
        int j = 0;
        int count;
        int m1 = -1, m2 = -1;

        /* Since there are 2n elements, median will
           be average of elements at index n-1 and
           n in the array obtained after merging ar1
           and ar2 */
        for (count = 0; count <= n; count++)
        {
            /* Below is to handle case where all
              elements of ar1[] are smaller than
              smallest(or first) element of ar2[] */
            if (i == n)
            {
                m1 = m2;
                m2 = ar2[0];
                break;
            }
```

```
            /* Below is to handle case where all
               elements of ar2[] are smaller than
               smallest(or first) element of ar1[] */
            else if (j == n)
            {
                m1 = m2;
                m2 = ar1[0];
                break;
            }

            if (ar1[i] < ar2[j])
            {
                /* Store the prev median */
                m1 = m2;
                m2 = ar1[i];
                i++;
            }
            else
            {
                /* Store the prev median */
                m1 = m2;
                m2 = ar2[j];
                j++;
            }
        }

        return (m1 + m2)/2;
    }

    /* Driver program to test above function */
    public static void main (String[] args)
    {
        int ar1[] = {1, 12, 15, 26, 38};
        int ar2[] = {2, 13, 17, 30, 45};

        int n1 = ar1.length;
        int n2 = ar2.length;
        if (n1 == n2)
            System.out.println("Median is " +
                        getMedian(ar1, ar2, n1));
        else
            System.out.println("arrays are of unequal size");
    }
}
```

Python3

```
 # A Simple Merge based O(n) Python 3 solution
# to find median of two sorted lists

# This function returns median of ar1[] and ar2[].
# Assumptions in this function:
# Both ar1[] and ar2[] are sorted arrays
# Both have n elements
def getMedian( ar1, ar2 , n):
    i = 0 # Current index of i/p list ar1[]

    j = 0 # Current index of i/p list ar2[]

    m1 = -1
    m2 = -1

    # Since there are 2n elements, median
    # will be average of elements at index
    # n-1 and n in the array obtained after
    # merging ar1 and ar2
    count = 0
    while count < n + 1:
        count += 1

        # Below is to handle case where all
        # elements of ar1[] are smaller than
        # smallest(or first) element of ar2[]
        if i == n:
            m1 = m2
            m2 = ar2[0]
            break

        # Below is to handle case where all
        # elements of ar2[] are smaller than
        # smallest(or first) element of ar1[]
        elif j == n:
            m1 = m2
            m2 = ar1[0]
            break
        if ar1[i] < ar2[j]:
            m1 = m2 # Store the prev median
            m2 = ar1[i]
            i += 1
        else:
            m1 = m2 # Store the prev median
            m2 = ar2[j]
            j += 1
    return (m1 + m2)/2
```

```
# Driver code to test above function
ar1 = [1, 12, 15, 26, 38]
ar2 = [2, 13, 17, 30, 45]
n1 = len(ar1)
n2 = len(ar2)
if n1 == n2:
    print("Median is ", getMedian(ar1, ar2, n1))
else:
    print("Doesn't work for arrays of unequal size")

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
 // A Simple Merge based O(n) solution
// to find median of two sorted arrays
using System;
class GFG
{
    // function to calculate median
    static int getMedian(int []ar1,
                         int []ar2,
                         int n)
    {
        int i = 0;
        int j = 0;
        int count;
        int m1 = -1, m2 = -1;

        // Since there are 2n elements,
        // median will be average of
        // elements at index n-1 and n in
        // the array obtained after
        // merging ar1 and ar2
        for (count = 0; count <= n; count++)
        {
            // Below is to handle case
            // where all elements of ar1[]
            // are smaller than smallest
            // (or first) element of ar2[]
            if (i == n)
            {
                m1 = m2;
                m2 = ar2[0];
                break;
            }

            /* Below is to handle case where all
```

```
            elements of ar2[] are smaller than
            smallest(or first) element of ar1[] */
            else if (j == n)
            {
                m1 = m2;
                m2 = ar1[0];
                break;
            }

            if (ar1[i] < ar2[j])
            {
                // Store the prev median
                m1 = m2;
                m2 = ar1[i];
                i++;
            }
            else
            {
                // Store the prev median
                m1 = m2;
                m2 = ar2[j];
                j++;
            }
        }

        return (m1 + m2)/2;
    }

    // Driver Code
    public static void Main ()
    {
        int []ar1 = {1, 12, 15, 26, 38};
        int []ar2 = {2, 13, 17, 30, 45};

        int n1 = ar1.Length;
        int n2 = ar2.Length;
        if (n1 == n2)
            Console.Write("Median is " +
                        getMedian(ar1, ar2, n1));
        else
            Console.Write("arrays are of unequal size");
    }
}
```

PHP

```
 <?php
// A Simple Merge based O(n) solution
```

```
// to find median of two sorted arrays

// This function returns median of
// ar1[] and ar2[]. Assumptions in
// this function: Both ar1[] and ar2[]
// are sorted arrays Both have n elements
function getMedian($ar1, $ar2, $n)
{
    // Current index of i/p array ar1[]
    $i = 0;

    // Current index of i/p array ar2[]
    $j = 0;
    $count;
    $m1 = -1; $m2 = -1;

    // Since there are 2n elements,
    // median will be average of elements
    // at index n-1 and n in the array
    // obtained after merging ar1 and ar2
    for ($count = 0; $count <= $n; $count++)
    {
        // Below is to handle case where
        // all elements of ar1[] are smaller
        // than smallest(or first) element of ar2[]
        if ($i == $n)
        {
            $m1 = $m2;
            $m2 = $ar2[0];
            break;
        }

        // Below is to handle case where all
        // elements of ar2[] are smaller than
        // smallest(or first) element of ar1[]
        else if ($j == $n)
        {
            $m1 = $m2;
            $m2 = $ar1[0];
            break;
        }

        if ($ar1[$i] < $ar2[$j])
        {
            // Store the prev median
            $m1 = $m2;
            $m2 = $ar1[$i];
            $i++;
```

```
        }
        else
        {
            // Store the prev median
            $m1 = $m2;
            $m2 = $ar2[$j];
            $j++;
        }
    }

    return ($m1 + $m2) / 2;
}

// Driver Code
$ar1 = array(1, 12, 15, 26, 38);
$ar2 = array(2, 13, 17, 30, 45);

$n1 = sizeof($ar1);
$n2 = sizeof($ar2);
if ($n1 == $n2)
    echo("Median is " .
          getMedian($ar1, $ar2, $n1));
else
    echo("Doesn't work for arrays".
         "of unequal size");

// This code is contributed by Ajit.
?>
```

Output :

```
Median is 16
```

Time Complexity : O(n)

Method 2 (By comparing the medians of two arrays)
This method works by first getting medians of the two sorted arrays and then comparing them.

Let ar1 and ar2 be the input arrays.

Algorithm :

```
1) Calculate the medians m1 and m2 of the input arrays ar1[]
   and ar2[] respectively.
2) If m1 and m2 both are equal then we are done.
     return m1 (or m2)
```

```
3) If m1 is greater than m2, then median is present in one
   of the below two subarrays.
    a)  From first element of ar1 to m1 (ar1[0...|_n/2_|])
    b)  From m2 to last element of ar2  (ar2[|_n/2_|...n-1])
4) If m2 is greater than m1, then median is present in one
   of the below two subarrays.
   a)  From m1 to last element of ar1  (ar1[|_n/2_|...n-1])
   b)  From first element of ar2 to m2 (ar2[0...|_n/2_|])
5) Repeat the above process until size of both the subarrays
   becomes 2.
6) If size of the two arrays is 2 then use below formula to get
  the median.
    Median = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2
```

Examples :

```
   ar1[] = {1, 12, 15, 26, 38}
   ar2[] = {2, 13, 17, 30, 45}
```

For above two arrays m1 = 15 and m2 = 17

For the above ar1[] and ar2[], m1 is smaller than m2. So median is present in one of the following two subarrays.

```
   [15, 26, 38] and [2, 13, 17]
```

Let us repeat the process for above two subarrays:

```
    m1 = 26 m2 = 13.
```

m1 is greater than m2. So the subarrays become

```
  [15, 26] and [13, 17]
Now size is 2, so median = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2
                       = (max(15, 13) + min(26, 17))/2
                       = (15 + 17)/2
                       = 16
```

Implementation :

C++

```
 // A divide and conquer based
// efficient solution to find
// median of two sorted arrays
// of same size.
#include<bits/stdc++.h>
using namespace std;

/* to get median of a
   sorted array */
int median(int [], int);

/* This function returns median
   of ar1[] and ar2[].
Assumptions in this function:
    Both ar1[] and ar2[] are
    sorted arrays
    Both have n elements */
int getMedian(int ar1[],
              int ar2[], int n)
{
    /* return -1 for
       invalid input */
    if (n <= 0)
        return -1;
    if (n == 1)
        return (ar1[0] +
                ar2[0]) / 2;
    if (n == 2)
        return (max(ar1[0], ar2[0]) +
                min(ar1[1], ar2[1])) / 2;

    /* get the median of
       the first array */
    int m1 = median(ar1, n);

    /* get the median of
       the second array */
    int m2 = median(ar2, n);

    /* If medians are equal then
       return either m1 or m2 */
    if (m1 == m2)
        return m1;

    /* if m1 < m2 then median must
       exist in ar1[m1....] and
                ar2[....m2] */
    if (m1 < m2)
```

```
    {
        if (n % 2 == 0)
            return getMedian(ar1 + n / 2 - 1,
                             ar2, n - n / 2 + 1);
        return getMedian(ar1 + n / 2,
                         ar2, n - n / 2);
    }

    /* if m1 > m2 then median must
       exist in ar1[....m1] and
                ar2[m2...] */
    if (n % 2 == 0)
        return getMedian(ar2 + n / 2 - 1,
                         ar1, n - n / 2 + 1);
    return getMedian(ar2 + n / 2,
                     ar1, n - n / 2);
}

/* Function to get median
   of a sorted array */
int median(int arr[], int n)
{
    if (n % 2 == 0)
        return (arr[n / 2] +
                arr[n / 2 - 1]) / 2;
    else
        return arr[n / 2];
}

// Driver code
int main()
{
    int ar1[] = {1, 2, 3, 6};
    int ar2[] = {4, 6, 8, 10};
    int n1 = sizeof(ar1) /
             sizeof(ar1[0]);
    int n2 = sizeof(ar2) /
             sizeof(ar2[0]);
    if (n1 == n2)
        cout << "Median is "
             << getMedian(ar1, ar2, n1);
    else
        cout << "Doesn't work for arrays "
             << "of unequal size";
    return 0;
}

// This code is contributed
```

```
// by Shivi_Aggarwal
```

C

```
 // A divide and conquer based efficient solution to find median
// of two sorted arrays of same size.
#include<bits/stdc++.h>
using namespace std;

int median(int [], int); /* to get median of a sorted array */

/* This function returns median of ar1[] and ar2[].
   Assumptions in this function:
   Both ar1[] and ar2[] are sorted arrays
   Both have n elements */
int getMedian(int ar1[], int ar2[], int n)
{
    /* return -1  for invalid input */
    if (n <= 0)
        return -1;
    if (n == 1)
        return (ar1[0] + ar2[0])/2;
    if (n == 2)
        return (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1])) / 2;

    int m1 = median(ar1, n); /* get the median of the first array */
    int m2 = median(ar2, n); /* get the median of the second array */

    /* If medians are equal then return either m1 or m2 */
    if (m1 == m2)
        return m1;

    /* if m1 < m2 then median must exist in ar1[m1....] and
        ar2[....m2] */
    if (m1 < m2)
    {
        if (n % 2 == 0)
            return getMedian(ar1 + n/2 - 1, ar2, n - n/2 +1);
        return getMedian(ar1 + n/2, ar2, n - n/2);
    }

    /* if m1 > m2 then median must exist in ar1[....m1] and
        ar2[m2...] */
    if (n % 2 == 0)
        return getMedian(ar2 + n/2 - 1, ar1, n - n/2 + 1);
    return getMedian(ar2 + n/2, ar1, n - n/2);
}
```

```
/* Function to get median of a sorted array */
int median(int arr[], int n)
{
    if (n%2 == 0)
        return (arr[n/2] + arr[n/2-1])/2;
    else
        return arr[n/2];
}

/* Driver program to test above function */
int main()
{
    int ar1[] = {1, 2, 3, 6};
    int ar2[] = {4, 6, 8, 10};
    int n1 = sizeof(ar1)/sizeof(ar1[0]);
    int n2 = sizeof(ar2)/sizeof(ar2[0]);
    if (n1 == n2)
        printf("Median is %d", getMedian(ar1, ar2, n1));
    else
        printf("Doesn't work for arrays of unequal size");
    return 0;
}
```

Output :

```
Median is 5
```

Time Complexity : O(logn)
Algorithmic Paradigm: Divide and Conquer

Median of two sorted arrays of different sizes

References:
http://en.wikipedia.org/wiki/Median

http://ocw.alfaisal.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/30C68118-E436-4FE3-8C79-6BAFBB07D935/0/ps9sol.pdf ds3etph5wn

Improved By : SaumyaBhatnagar, jit_t, nitin mittal, Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/median-of-two-sorted-arrays/

Chapter 168

Median of two sorted arrays with different sizes in O(log(min(n, m)))

Median of two sorted arrays with different sizes in O(log(min(n, m))) - GeeksforGeeks

Given two sorted arrays, a[] and b[], task is to find the median of these sorted arrays, in O(log(min(n, m)), when n is the number of elements in the first array, and m is the number of elements in the second array.

Prerequisite : Median of two different sized sorted arrays.

Examples :

```
Input : ar1[] = {-5, 3, 6, 12, 15}
        ar2[] = {-12, -10, -6, -3, 4, 10}
        The merged array is :
        ar3[] = {-12, -10, -6, -5 , -3,
                 3, 4, 6, 10, 12, 15}
Output : The median is 3.

Input : ar1[] = {2, 3, 5, 8}
        ar2[] = {10, 12, 14, 16, 18, 20}
        The merged array is :
        ar3[] = {2, 3, 5, 8, 10, 12, 14, 16, 18, 20}
        if the number of the elements are even,
        so there are two middle elements,
        take the average between the two :
        (10 + 12) / 2 = 11.
Output : The median is 11.
```

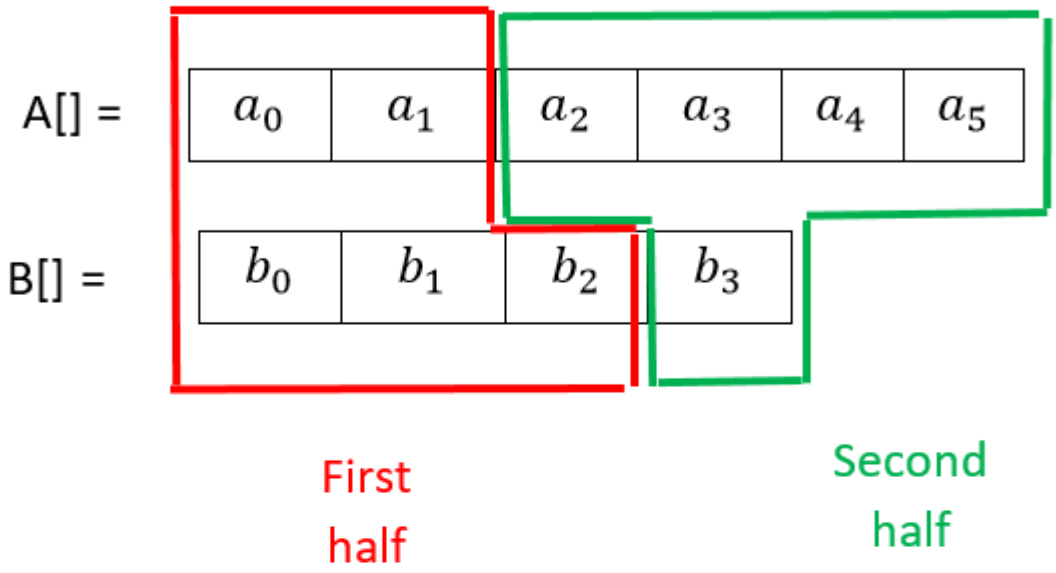
Note : In case of even numbers in total and if we want to return a median that exist in the merged array we can return the element in the (n+m)/2 or (n+m)/2 – 1 position. In that case the median can be 10 or 12.

Approach : Start partitioning the two arrays into two groups of halves (not two parts, but both partitioned should have same number of elements). The first half contains some first elements from the first and the second arrays, and the second half contains the rest (or the last) elements form the first and the second arrays. Because the arrays can be of different sizes, it does not mean to take every half from each array. The below example clarifies the explanation. Reach a condition such that, every element in the first half is less than or equal to every element in the second half.

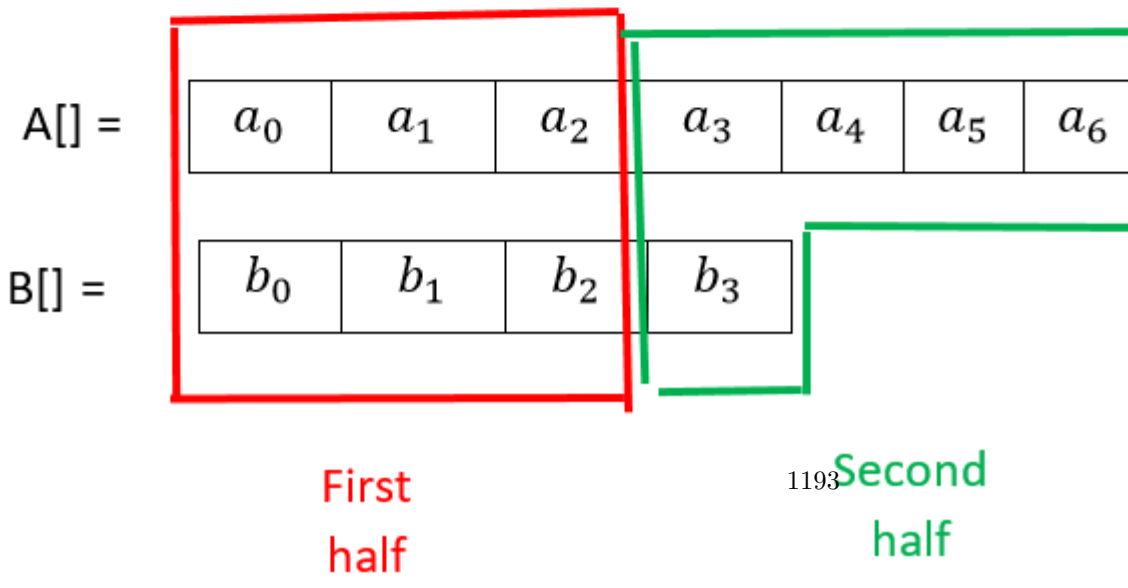
How to reach this condition ?
Example in the case of even numbers. Suppose, partition is found. Because A[] and B[] are two sorted arrays, a1 is less than or equal to a2, and b2 is less than or equal to b3. Now, to check if a1 is less than or equal to b3, and if b2 is less than or equal to a2. If that's the case, it means that every element in the first half is less than or equal to every element in the second half, because, a1 is greater than or equal to every element before it (a0) in A[], and b2 is greater than or equal to every element before it (b1 and b0) in B[]. In case of even numbers in total the median will be the average between max of a1, b2 and the min of a2, b3, but in case of odd numbers in total the median will be the max of a2, b2. But if it is not these two cases, there are two options (in referring to the even numbers example) : b2 > a2 or a1 > b3
if, b2 > a2 it means that, search on the right side of the array, and if a1 > b3 it means that, search on the left side of the array, until desired condition is found.

Suppose A[] and B[] are two sorted arrays:

Case of even numbers:

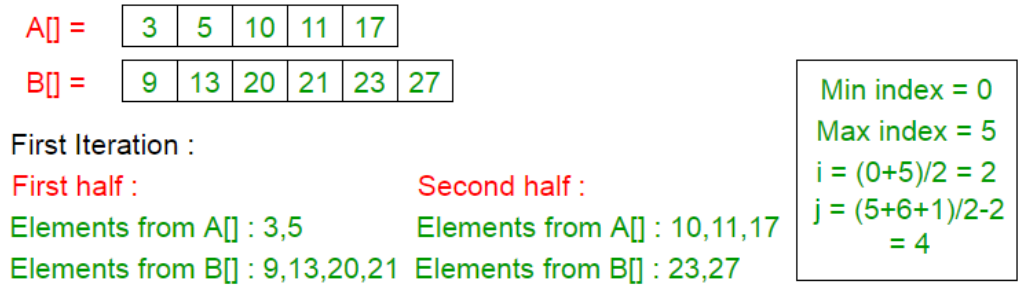


Case of odd numbers (with an extra element in the first half):

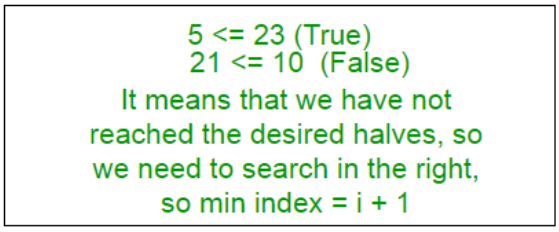


Why the above condition leads to the median ?
The median is the (n + 1) / 2 smallest element of the array, and here, the median is the (n + m + 1) / 2 smallest element among the two arrays. If, all the elements in the first half are less than (or equal) to all elements in the second half, in case of odd numbers in total, just calculate the maximum between the last two elements in the first half (a2 and b2 in our example),and this will lead us to the (n + m + 1) / 2 smallest element among the two arrays, which is the median ((7 + 4 + 1) / 2 = 6). But in case of even numbers in total, calculate the average between the maximum of the last two elements in the first half (a1 and b2 in our example) with its successive number among the arrays which is the minimum of first two elements in the second half (a2 and b3 in our example).

The process of the partition :
To make two halves, make the partition such that the index that partitioning array A[] + the index that partitioning array B[] are equal to the total number of elements plus one divided by 2, i.e. (n + m + 1) / 2 (+1 is, if the total number of elements is odd).
First, define two variables : min_index and max_index, and initialize min_index to 0, and max_index to the length of the smaller array. In these below examples A[] is the smaller array.
To partition A[], use the formula (min_index + max_index) / 2 and insert it to a variable i. To partition B[], use the formula (n + m + 1) / 2 – i and insert it to a variable j.
the variable i means the number of elements to be inserted from A[] into the first half, and j means the number of elements to be inserted from B[] into the first half, the rest of the elements will be inserted into the second half.
Take a look at the below examples :
Example 1 :



When the number of elements is odd, we have an extra element in the first half :



Second Iteration :

First half :

Elements from A[] : 3,5,10,11

Elements from B[] : 9,13

Second half :

Elements from A[] : 17

Elements from B[] : 20,21,23,27

Min index = 3
Max index = 5
i = (3+5)/2 = 4
j = (5+6+1)/2-4 = 2

11 <= 20 (True)
13 <= 17 (True)
we have reached the desired halves, so we returning the max(11,13) which is the (n+m+1)/2 smallest element, i.e, the median if the two arrays

Example 2 (This example refers to the condition that returns a median that exists in the merged array) **:**

A[] = | 2 | 3 | 5 | 8 |

B[] = | 10 | 12 | 14 | 16 | 18 | 20 |

First Iteration :

First half :

Elements from A[] : 2,3

Elements from B[] : 10,12,14

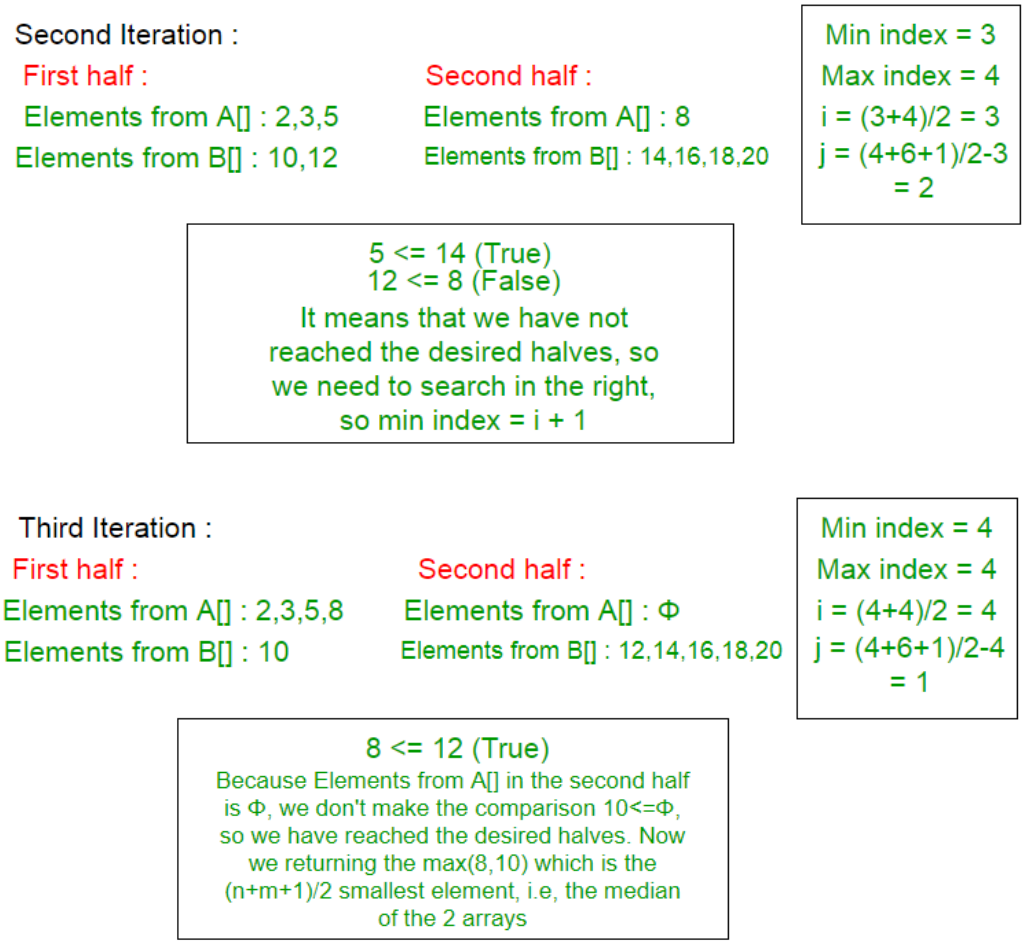
Second half :

Elements from A[] : 5,8

Elements from B[] : 16,18,20

Min index = 0
Max index = 4
i = (0+4)/2 = 2
j = (4+6+1)/2-2 = 3

3 <= 16 (True)
14 <= 5 (False)
It means that we have not reached the desired halves, so we need to search in the right, so min index = i + 1



Below is the implementation of above approach :

C++

```
 // CPP code for median with case of returning
// double value when even number of elements are
// present in both array combinely
#include<bits/stdc++.h>
using std::cout;

int maximum(int a, int b);
int minimum(int a, int b);

// Function to find median of two sorted arrays
```

```
double findMedianSortedArrays(int *a, int n,
                              int *b, int m)
{

    int min_index = 0, max_index = n, i, j, median;

    while (min_index <= max_index)
    {
        i = (min_index + max_index) / 2;
        j = ((n + m + 1) / 2) - i;

        // if i = n, it means that Elements from a[] in
        // the second half is an empty set. and if j = 0,
        // it means that Elements from b[] in the first
        // half is an empty set. so it is necessary to
        // check that, because we compare elements from
        // these two groups.
        // Searching on right
        if (i < n && j > 0 && b[j - 1] > a[i])
            min_index = i + 1;

        // if i = 0, it means that Elements from a[] in
        // the first half is an empty set and if j = m,
        // it means that Elements from b[] in the second
        // half is an empty set. so it is necessary to
        // check that, because we compare elements
        // from these two groups.
        // searching on left
        else if (i > 0 && j < m && b[j] < a[i - 1])
            max_index = i - 1;

        // we have found the desired halves.
        else
        {
            // this condition happens when we don't have any
            // elements in the first half from a[] so we
            // returning the last element in b[] from
            // the first half.
            if (i == 0)
                median = b[j - 1];

            // and this condition happens when we don't
            // have any elements in the first half from
            // b[] so we returning the last element in
            // a[] from the first half.
            else if (j == 0)
                median = a[i - 1];
            else
```

```
                median = maximum(a[i - 1], b[j - 1]);
            break;
        }
    }

    // calculating the median.
    // If number of elements is odd there is
    // one middle element.
    if ((n + m) % 2 == 1)
        return (double)median;

    // Elements from a[] in the second half is an empty set.
    if (i == n)
        return (median+b[j]) / 2.0;

    // Elements from b[] in the second half is an empty set.
    if (j == m)
        return (median + a[i]) / 2.0;

    return (median + minimum(a[i], b[j])) / 2.0;
}

// Function to find max
int maximum(int a, int b)
{
    return a > b ? a : b;
}

// Function to find minimum
int minimum(int a, int b)
{
    return a < b ? a : b;
}

// Driver code
int main()
{
    int a[] = {900};
    int b[] = { 10, 13, 14};
    int n = sizeof(a) / sizeof(int);
    int m = sizeof(b) / sizeof(int);

    // we need to define the smaller array as the
    // first parameter to make sure that the
    // time complexity will be O(log(min(n,m)))
    if (n < m)
        cout << "The median is : "
             << findMedianSortedArrays(a, n, b, m);
```

```
    else
        cout << "The median is : "
             << findMedianSortedArrays(b, m, a, n);

    return 0;
}
```

Java

```
 // Java code for median with
// case of returning double
// value when even number of
// elements are present in
// both array combinely
import java.io.*;

class GFG
{
    static int []a = new int[]{900};
    static int []b = new int[]{10, 13, 14};

    // Function to find max
    static int maximum(int a, int b)
    {
        return a > b ? a : b;
    }

    // Function to find minimum
    static int minimum(int a, int b)
    {
        return a < b ? a : b;
    }

    // Function to find median
    // of two sorted arrays
    static double findMedianSortedArrays(int n,
                                         int m)
    {

        int min_index = 0,
            max_index = n, i = 0,
            j = 0, median = 0;

        while (min_index <= max_index)
        {
            i = (min_index + max_index) / 2;
            j = ((n + m + 1) / 2) - i;
```

```
        // if i = n, it means that Elements
        // from a[] in the second half is an
        // empty set. and if j = 0, it means
        // that Elements from b[] in the first
        // half is an empty set. so it is
        // necessary to check that, because we
        // compare elements from these two
        // groups. Searching on right
        if (i < n && j > 0 && b[j - 1] > a[i])
            min_index = i + 1;

        // if i = 0, it means that Elements
        // from a[] in the first half is an
        // empty set and if j = m, it means
        // that Elements from b[] in the second
        // half is an empty set. so it is
        // necessary to check that, because
        // we compare elements from these two
        // groups. searching on left
        else if (i > 0 && j < m && b[j] < a[i - 1])
            max_index = i - 1;

        // we have found the desired halves.
        else
        {
            // this condition happens when we
            // don't have any elements in the
            // first half from a[] so we
            // returning the last element in
            // b[] from the first half.
            if (i == 0)
                median = b[j - 1];

            // and this condition happens when
            // we don't have any elements in the
            // first half from b[] so we
            // returning the last element in
            // a[] from the first half.
            else if (j == 0)
                median = a[i - 1];
            else
                median = maximum(a[i - 1],
                                 b[j - 1]);
            break;
        }
    }

    // calculating the median.
```

```
        // If number of elements is odd
        // there is one middle element.
        if ((n + m) % 2 == 1)
            return (double)median;

        // Elements from a[] in the
        // second half is an empty set.
        if (i == n)
            return (median + b[j]) / 2.0;

        // Elements from b[] in the
        // second half is an empty set.
        if (j == m)
            return (median + a[i]) / 2.0;

        return (median + minimum(a[i],
                                 b[j])) / 2.0;
    }

    // Driver code
    public static void main(String args[])
    {
        int n = a.length;
        int m = b.length;

        // we need to define the
        // smaller array as the
        // first parameter to
        // make sure that the
        // time complexity will
        // be O(log(min(n,m)))
        if (n < m)
            System.out.print("The median is : " +
                   findMedianSortedArrays(n, m));
        else
            System.out.print("The median is : " +
                   findMedianSortedArrays(m, n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Python3

```
 # Python code for median with
# case of returning double
# value when even number
```

```
# of elements are present
# in both array combinely
median = 0
i = 0
j = 0

# def to find max
def maximum(a, b) :
    return a if a > b else b

# def to find minimum
def minimum(a, b) :
    return a if a < b else b

# def to find median
# of two sorted arrays
def findMedianSortedArrays(a, n, b, m) :

    global median, i, j
    min_index = 0
    max_index = n

    while (min_index <= max_index) :

        i = int((min_index + max_index) / 2)
        j = int(((n + m + 1) / 2) - i)

        # if i = n, it means that
        # Elements from a[] in the
        # second half is an empty
        # set. and if j = 0, it
        # means that Elements from
        # b[] in the first half is
        # an empty set. so it is
        # necessary to check that,
        # because we compare elements
        # from these two groups.
        # Searching on right
        if (i < n and j > 0 and b[j - 1] > a[i]) :
            min_index = i + 1

        # if i = 0, it means that
        # Elements from a[] in the
        # first half is an empty
        # set and if j = m, it means
        # that Elements from b[] in
        # the second half is an empty
        # set. so it is necessary to
```

```
    # check that, because we compare
    # elements from these two groups.
    # searching on left
    elif (i > 0 and j < m and b[j] < a[i - 1]) :
        max_index = i - 1

    # we have found the
    # desired halves.
    else :

        # this condition happens when
        # we don't have any elements
        # in the first half from a[]
        # so we returning the last
        # element in b[] from the
        # first half.
        if (i == 0) :
            median = b[j - 1]

        # and this condition happens
        # when we don't have any
        # elements in the first half
        # from b[] so we returning the
        # last element in a[] from the
        # first half.
        elif (j == 0) :
            median = a[i - 1]
        else :
            median = maximum(a[i - 1], b[j - 1])
        break



# calculating the median.
# If number of elements
# is odd there is
# one middle element.

if ((n + m) % 2 == 1) :
    return median

# Elements from a[] in the
# second half is an empty set.
if (i == n) :
    return ((median + b[j]) / 2.0)

# Elements from b[] in the
# second half is an empty set.
```

```
    if (j == m) :
        return ((median + a[i]) / 2.0)

    return ((median + minimum(a[i], b[j])) / 2.0)


# Driver code
a = [900]
b = [10, 13, 14]
n = len(a)
m = len(b)

# we need to define the
# smaller array as the
# first parameter to make
# sure that the time complexity
# will be O(log(min(n,m)))
if (n < m) :
    print ("The median is : {}".format(findMedianSortedArrays(a, n, b, m)))
else :
    echo ("The median is : {}".format(findMedianSortedArrays(b, m, a, n)))

# This code is contributed
# by Manish Shaw(manishshaw1)
```

C#

```
 // C# code for median with case of returning
// double value when even number of elements
// are present in both array combinely
using System;

class GFG {

    // Function to find max
    static int maximum(int a, int b)
    {
        return a > b ? a : b;
    }

    // Function to find minimum
    static int minimum(int a, int b)
    {
        return a < b ? a : b;
    }

    // Function to find median of two sorted
    // arrays
```

```
static double findMedianSortedArrays(ref int []a,
                      int n, ref int []b, int m)
{

    int min_index = 0, max_index = n, i = 0,
    j = 0, median = 0;

    while (min_index <= max_index)
    {
        i = (min_index + max_index) / 2;
        j = ((n + m + 1) / 2) - i;

        // if i = n, it means that Elements
        // from a[] in the second half is an
        // empty set. and if j = 0, it means
        // that Elements from b[] in the first
        // half is an empty set. so it is
        // necessary to check that, because we
        // compare elements from these two
        // groups. Searching on right
        if (i < n && j > 0 && b[j - 1] > a[i])
            min_index = i + 1;

        // if i = 0, it means that Elements
        // from a[] in the first half is an
        // empty set and if j = m, it means
        // that Elements from b[] in the second
        // half is an empty set. so it is
        // necessary to check that, because
        // we compare elements from these two
        // groups. searching on left
        else if (i > 0 && j < m && b[j] < a[i - 1])
            max_index = i - 1;

        // we have found the desired halves.
        else
        {
            // this condition happens when we
            // don't have any elements in the
            // first half from a[] so we
            // returning the last element in
            // b[] from the first half.
            if (i == 0)
                median = b[j - 1];

            // and this condition happens when
            // we don't have any elements in the
            // first half from b[] so we
```

```
            // returning the last element in
            // a[] from the first half.
            else if (j == 0)
                median = a[i - 1];
            else
                median = maximum(a[i - 1], b[j - 1]);
            break;
        }
    }

    // calculating the median.
    // If number of elements is odd
    // there is one middle element.
    if ((n + m) % 2 == 1)
        return (double)median;

    // Elements from a[] in the second
    // half is an empty set.
    if (i == n)
        return (median+b[j]) / 2.0;

    // Elements from b[] in the second
    // half is an empty set.
    if (j == m)
        return (median + a[i]) / 2.0;

    return (median + minimum(a[i], b[j])) / 2.0;
}

// Driver code
static void Main()
{
    int []a = new int[]{900};
    int []b = new int[]{ 10, 13, 14};
    int n = a.Length;
    int m = b.Length;

    // we need to define the smaller
    // array as the first parameter to
    // make sure that the time
    // complexity will be O(log(min(n,m)))
    if (n < m)
        Console.Write("The median is : "
        + findMedianSortedArrays(ref a, n,
                                 ref b, m));
    else
        Console.Write("The median is : "
        + findMedianSortedArrays(ref b, m,
```

```
                                    ref a, n));
    }
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

PHP

```
 <?php
// PHP code for median with
// case of returning double
// value when even number
// of elements are present
// in both array combinely
$median = 0;
$i = 0; $j = 0;

// Function to find max
function maximum($a, $b)
{
    return $a > $b ? $a : $b;
}

// Function to find minimum
function minimum($a, $b)
{
    return $a < $b ? $a : $b;
}

// Function to find median
// of two sorted arrays
function findMedianSortedArrays(&$a, $n,
                                &$b, $m)
{
    global $median, $i, $j;
    $min_index = 0;
    $max_index = $n;

    while ($min_index <= $max_index)
    {
        $i = intval(($min_index +
                     $max_index) / 2);
        $j = intval((($n + $m + 1) /
                           2) - $i);

        // if i = n, it means that
        // Elements from a[] in the
```

```
// second half is an empty
// set. and if j = 0, it
// means that Elements from
// b[] in the first half is
// an empty set. so it is
// necessary to check that,
// because we compare elements
// from these two groups.
// Searching on right
if ($i < $n && $j > 0 &&
    $b[$j - 1] > $a[$i])
    $min_index = $i + 1;

// if i = 0, it means that
// Elements from a[] in the
// first half is an empty
// set and if j = m, it means
// that Elements from b[] in
// the second half is an empty
// set. so it is necessary to
// check that, because we compare
// elements from these two groups.
// searching on left
else if ($i > 0 && $j < $m &&
         $b[$j] < $a[$i - 1])
    $max_index = $i - 1;

// we have found the
// desired halves.
else
{
    // this condition happens when
    // we don't have any elements
    // in the first half from a[]
    // so we returning the last
    // element in b[] from the
    // first half.
    if ($i == 0)
        $median = $b[$j - 1];

    // and this condition happens
    // when we don't have any
    // elements in the first half
    // from b[] so we returning the
    // last element in a[] from the
    // first half.
    else if ($j == 0)
        $median = $a[$i - 1];
```

```
            else
                $median = maximum($a[$i - 1],
                                  $b[$j - 1]);
            break;
        }
    }

    // calculating the median.
    // If number of elements
    // is odd there is
    // one middle element.

    if (($n + $m) % 2 == 1)
        return $median;

    // Elements from a[] in the
    // second half is an empty set.
    if ($i == $n)
        return (($median +
                 $b[$j]) / 2.0);

    // Elements from b[] in the
    // second half is an empty set.
    if ($j == $m)
        return (($median +
                 $a[$i]) / 2.0);

    return (($median +
             minimum($a[$i],
                     $b[$j])) / 2.0);
}

// Driver code
$a = array(900);
$b = array(10, 13, 14);
$n = count($a);
$m = count($b);

// we need to define the
// smaller array as the
// first parameter to make
// sure that the time complexity
// will be O(log(min(n,m)))
if ($n < $m)
    echo ("The median is : " .
           findMedianSortedArrays($a, $n,
                                  $b, $m));
else
```

```
    echo ("The median is : " .
           findMedianSortedArrays($b, $m,
                                  $a, $n));

// This code is contributed
// by Manish Shaw(manishshaw1)
?>
```

Output:

```
The median is : 13.5
```

Another Approach : Same program, but returns the median that exist in the merged array((n + m) / 2 – 1 position):

C++

```
 // CPP code for finding median of the given two
// sorted arrays that exists in the merged array ((n+m) / 2 - 1 position)
#include<bits/stdc++.h>
using std::cout;

int maximum(int a, int b);

// Function to find median of given two sorted arrays
int findMedianSortedArrays(int *a, int n,
                           int *b, int m)
{

    int min_index = 0, max_index = n, i, j;

    while (min_index <= max_index)
    {
        i = (min_index + max_index) / 2;
        j = ((n + m + 1) / 2) - i;

        // if i = n, it means that Elements from a[] in
        // the second half is an empty set. If j = 0, it
        // means that Elements from b[] in the first half
        // is an empty set. so it is necessary to check that,
        // because we compare elements from these two groups.
        // searching on right
        if (i < n && j > 0 && b[j - 1] > a[i])
            min_index = i + 1;

        // if i = 0, it means that Elements from a[] in the
        // first half is an empty set and if j = m, it means
```

```
        // that Elements from b[] in the second half is an
        // empty set. so it is necessary to check that,
        // because we compare elements from these two groups.
        // searching on left
        else if (i > 0 && j < m && b[j] < a[i - 1])
            max_index = i - 1;

        // we have found the desired halves.
        else
        {
            // this condition happens when we don't have
            // any elements in the first half from a[] so
            // we returning the last element in b[] from
            // the first half.
            if (i == 0)
                return b[j - 1];

            // and this condition happens when we don't have any
            // elements in the first half from b[] so we
            // returning the last element in a[] from the first half.
            if (j == 0)
                return a[i - 1];
            else
                return maximum(a[i - 1], b[j - 1]);
        }
    }

    cout << "ERROR!!! " << "returning\n";
    return 0;
}

// Function to find maximum
int maximum(int a, int b)
{
    return a > b ? a : b;
}

// Driver code
int main()
{
    int a[] = {900};
    int b[] = { 10,13,14};
    int n = sizeof(a) / sizeof(int);
    int m = sizeof(b) / sizeof(int);

    // we need to define the smaller array as the first
    // parameter to make sure that the time complexity
    // will be O(log(min(n,m)))
```

```
    if (n < m)
        cout << "The median is: "
             << findMedianSortedArrays(a, n, b, m);
    else
        cout << "The median is: "
             << findMedianSortedArrays(b, m, a, n);
    return 0;
}
```

C#

```
 // C# code for finding median
// of the given two sorted
// arrays that exists in the
// merged array ((n+m) / 2 -
// 1 position)
using System;

class GFG
{
    // Function to find maximum
    static int maximum(int a,
                       int b)
    {
        return a > b ? a : b;
    }

    // Function to find median
    // of given two sorted arrays
    static int findMedianSortedArrays(ref int []a, int n,
                                      ref int []b, int m)
    {

        int min_index = 0,
            max_index = n, i, j;

        while (min_index <= max_index)
        {
            i = (min_index + max_index) / 2;
            j = ((n + m + 1) / 2) - i;

            // if i = n, it means that
            // Elements from a[] in the
            // second half is an empty
            // set. If j = 0, it means
            // that Elements from b[]
            // in the first half is an
            // empty set. so it is
```

```
// necessary to check that,
// because we compare elements
// from these two groups.
// searching on right
if (i < n && j > 0 &&
    b[j - 1] > a[i])
    min_index = i + 1;

// if i = 0, it means that
// Elements from a[] in the
// first half is an empty set
// and if j = m, it means that
// Elements from b[] in the
// second half is an empty set.
// so it is necessary to check
// that, because we compare
// elements from these two
// groups. searching on left
else if (i > 0 && j < m &&
         b[j] < a[i - 1])
    max_index = i - 1;

// we have found the
// desired halves.
else
{
    // this condition happens
    // when we don't have any
    // elements in the first
    // half from a[] so we
    // returning the last
    // element in b[] from
    // the first half.
    if (i == 0)
        return b[j - 1];

    // and this condition happens
    // when we don't have any
    // elements in the first half
    // from b[] so we returning the
    // last element in a[] from the
    // first half.
    if (j == 0)
        return a[i - 1];
    else
        return maximum(a[i - 1],
                       b[j - 1]);
}
```

```
        }

        Console.Write ("ERROR!!! " +
                       "returning\n");
        return 0;
    }

    // Driver code
    static void Main()
    {
        int []a = new int[]{900};
        int []b = new int[]{10, 13, 14};
        int n = a.Length;
        int m = b.Length;

        // we need to define the smaller
        // array as the first parameter
        // to make sure that the time
        // complexity will be O(log(min(n,m)))
        if (n < m)
            Console.Write ("The median is: " +
                           findMedianSortedArrays(ref a, n,
                                                  ref b, m));
        else
            Console.Write ("The median is: " +
                           findMedianSortedArrays(ref b, m,
                                                  ref a, n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

```
The median is: 13
```

Time Complexity : O(log(min(n, m))), where n and m are the sizes of given sorted array

Improved By : Shlomi Elhaiani, manishshaw1

Source

https://www.geeksforgeeks.org/median-two-sorted-arrays-different-sizes-ologminn-m/

Chapter 169

Middle of three using minimum comparisons

Middle of three using minimum comparisons - GeeksforGeeks

Given three distinct numbers a, b and c find the number with a value in middle.

Examples :

```
Input : a = 20, b = 30, c = 40
Output : 30

Input : a = 12, n = 32, c = 11
Output : 12
```

Simple Approach :

C++

```
 // CPP program to find  middle of three distinct
// numbers
#include <bits/stdc++.h>
using namespace std;

int middleOfThree(int a, int b, int c)
{
    // Checking for b
    if ((a < b && b < c) || (c < b && b < a))
       return b;

    // Checking for a
```

```
    else if ((b < a && a < c) || (c < a && a < b))
       return a;

    else
       return c;
}

// Driver Code
int main()
{
    int a = 20, b = 30, c = 40;
    cout << middleOfThree(a, b, c);
    return 0;
}
```

Java

```
 // Java program to find middle of
// three distinct numbers
import java.util.*;

class Middle
{
    // Function to find the middle of three number
    public static int middleOfThree(int a, int b,
                                        int c)
    {
        // Checking for b
        if ((a < b && b < c) || (c < b && b < a))
            return b;

        // Checking for a
        else if ((b < a && a < c) || (c < a && a < b))
        return a;

        else
        return c;
    }

    // driver code
    public static void main(String[] args)
    {
        int a = 20, b = 30, c = 40;
        System.out.println( middleOfThree(a, b, c) );
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
 # Python3 program to find middle
# of three distinct numbers

def middleOfThree(a, b, c):

    # Checking for b
    if ((a < b and b < c) or (c < b and b < a)) :
        return b;

    # Checking for a
    if ((b < a and a < c) or (c < a and a < b)) :
        return a;

    else :
        return c

# Driver Code
a = 20
b = 30
c = 40
print(middleOfThree(a, b, c))

# This code is contributed by rishabh_jain
```

C#

```
 // C# program to find middle of
// three distinct numbers
using System;

class Middle
{
    // Function to find the middle of three number
    public static int middleOfThree(int a, int b,
                                     int c)
    {
        // Checking for b
        if ((a < b && b < c) || (c < b && b < a))
            return b;

        // Checking for a
        else if ((b < a && a < c) || (c < a && a < b))
        return a;

        else
```

```
        return c;
    }

    // Driver code
    public static void Main()
    {
        int a = 20, b = 30, c = 40;
        Console.WriteLine( middleOfThree(a, b, c) );
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP program to find middle
// of three distinct numbers

function middleOfThree($a, $b, $c)
{

    // Checking for b
    if (($a < $b && $b < $c) or
        ($c < $b && $b < $a))
    return $b;

    // Checking for a
    else if (($b < $a and $a < $c) or
             ($c < $a and $a < $b))
    return $a;

    else
    return $c;
}

    // Driver Code
    $a = 20;
    $b = 30;
    $c = 40;
    echo middleOfThree($a, $b, $c);

// This code is contributed by anuj_67.
?>
```

Output :

30

But approach used above is not efficient because of extra comparisons, which can be easily minimized. In case first part is false, it will execute remaining half to check the condition. This problem remains same if we are checking for **a** also.

Better Approach (Needs less comparison):

C++

```
 // CPP program to find middle of three distinct
// numbers
#include <bits/stdc++.h>
using namespace std;

// Function to find the middle of three numbers
int middleOfThree(int a, int b, int c)
{
    // Compare each three number to find middle
    // number. Enter only if a > b
    if (a > b)
    {
        if (b > c)
            return b;
        else if (a > c)
            return c;
        else
            return a;
    }
    else
    {
        // Decided a is not greater than b.
        if (a > c)
            return a;
        else if (b > c)
            return c;
        else
            return b;
    }
}

// Driver Code
int main()
{
    int a = 20, b = 30, c = 40;
    cout << middleOfThree(a, b, c);
    return 0;
}
```

Java

```
 // Java program to find middle of
// three distinct numbers
import java.util.*;

class Middle
{
    // Function to find the middle of three number
    public static int middleOfThree(int a, int b,
                                            int c)
    {
        // Compare each three number to find
        // middle number. Enter only if a > b
        if (a > b)
        {
            if (b > c)
                return b;
            else if (a > c)
                return c;
            else
                return a;
        }
        else
        {
            // Decided a is not greater than b.
            if (a > c)
                return a;
            else if (b > c)
                return c;
            else
                return b;
        }
    }

    // driver code
    public static void main(String[] args)
    {
        int a = 20, b = 30, c = 40;
        System.out.println(middleOfThree(a, b, c));
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
 # Python3 program to find middle
```

```
# of three distinct numbers

# Function to find the middle of three numbers
def middleOfThree(a, b, c) :

    # Compare each three number to find
    # middle number. Enter only if a > b
    if a > b :
        if (b > c):
            return b
        elif (a > c) :
            return c
        else :
            return a
    else:
        # Decided a is not greater than b.
        if (a > c) :
            return a
        elif (b > c) :
            return c
        else :
            return b

# Driver Code
a = 20
b = 30
c = 40
print( middleOfThree(a, b, c) )

# This code is contributed by rishabh_jain
```

C#

```
 // C# program to find middle of
// three distinct numbers
using System;

class Middle
{
    // Function to find the middle of three number
    public static int middleOfThree(int a, int b,
                                            int c)
    {
        // Compare each three number to find
        // middle number. Enter only if a > b
        if (a > b)
        {
            if (b > c)
```

```
                return b;
            else if (a > c)
                return c;
            else
                return a;
        }
        else
        {
            // Decided a is not greater than b.
            if (a > c)
                return a;
            else if (b > c)
                return c;
            else
                return b;
        }
    }

    // Driver code
    public static void Main()
    {
        int a = 20, b = 30, c = 40;
        Console.WriteLine(middleOfThree(a, b, c));
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP program to find middle
// of three distinct numbers

// Function to find the middle
// of three numbers
function middleOfThree( $a, $b, $c)
{

    // Compare each three number
    // to find middle number.
    // Enter only if a > b
    if ($a > $b)
    {
        if ($b > $c)
            return $b;
        else if ($a > $c)
            return $c;
```

```
        else
            return $a;
    }
    else
    {

        // Decided a is not
        // greater than b.
        if ($a > $c)
            return $a;
        else if ($b > $c)
            return $c;
        else
            return $b;
    }
}

    // Driver Code
    $a = 20; $b = 30;
    $c = 40;
    echo middleOfThree($a, $b, $c);

// This code is contributed by anuj_67.
?>
```

Output:

```
30
```

This approach is efficient and having less number of comparisons. Outer IF loop will only be executed if a > b otherwise, outer ELSE will be executed.
Other Efficient Approach :

C++

```
 // CPP program to find middle of three distinct
// numbers
#include <bits/stdc++.h>
using namespace std;

// Function to find the middle of three number
int middleOfThree(int a, int b, int c)
{
    // x is positive if a is greater than b.
    // x is negative if b is greater than a.
    int x = a - b;
```

```
    int y = b - c;  // Similar to x
    int z = a - c;  // Similar to x and y.

    // Checking if b is middle (x and y both
    // are positive)
    if (x * y > 0)
        return b;

    // Checking if c is middle (x and z both
    // are positive)
    else if (x * z > 0)
        return c;
    else
        return a;
}

// Driver Code
int main()
{
    int a = 20, b = 30, c = 40;
    cout << middleOfThree(a, b, c);
    return 0;
}
```

Java

```
 //java program to find middle of three distinct
// numbers
import java.util.*;

class Middle
{
    // Function to find the middle of three number
    public static int middleOfThree(int a, int b,
                                            int c)
    {
        // x is positive if a is greater than b.
        // x is negative if b is greater than a.
        int x = a - b;

        int y = b - c; // Similar to x
        int z = a - c; // Similar to x and y.

        // Checking if b is middle (x and y
        // both are positive)
        if (x * y > 0)
            return b;
```

```
        // Checking if c is middle (x and z
        // both are positive)
        else if (x * z > 0)
            return c;
        else
            return a;
    }

    // driver code
    public static void main(String[] args)
    {
        int a = 20, b = 30, c = 40;
        System.out.println( middleOfThree(a, b, c) );
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
 # Python3 program to find middle
# of three distinct numbers

# Function to find the middle of three number
def middleOfThree(a, b, c) :

    # x is positive if a is greater than b.
    # x is negative if b is greater than a.
    x = a - b

    # Similar to x
    y = b - c

    # Similar to x and y.
    z = a - c

    # Checking if b is middle (x and y
    # both are positive)
    if x * y > 0:
        return b

    # Checking if c is middle (x and z
    # both are positive)
    elif (x * z > 0) :
        return
    else :
        return a
```

```
# Driver Code
a = 20
b = 30
c = 40
print(middleOfThree(a, b, c))

# This code is contributed by rishabh_jain
```

C#

```
 //C# program to find middle of three distinct
// numbers
using System;

class Middle
{
    // Function to find the middle of three number
    public static int middleOfThree(int a, int b,
                                     int c)
    {
        // x is positive if a is greater than b.
        // x is negative if b is greater than a.
        int x = a - b;

        // Similar to x
        int y = b - c;

         // Similar to x and y.
        int z = a - c;

        // Checking if b is middle (x and y
        // both are positive)
        if (x * y > 0)
            return b;

        // Checking if c is middle (x and z
        // both are positive)
        else if (x * z > 0)
            return c;
        else
            return a;
    }

    // Driver code
    public static void Main()
    {
        int a = 20, b = 30, c = 40;
        Console.WriteLine( middleOfThree(a, b, c) );
```

```
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP program to find middle
// of three distinct numbers

// Function to find the
// middle of three number
function middleOfThree($a, $b, $c)
{

    // x is positive if a
    // is greater than b.
    // x is negative if b
    // is greater than a.
    $x = $a - $b;

    // Similar to x
    $y = $b - $c;

    // Similar to x and y.
    $z = $a - $c;

    // Checking if b is
    // middle (x and y both
    // are positive)
    if ($x * $y > 0)
        return $b;

    // Checking if c is
    // middle (x and z both
    // are positive)
    else if ($x * $z > 0)
        return $c;
    else
        return $a;
}

    // Driver Code
    $a = 20;
    $b = 30;
    $c = 40;
    echo middleOfThree($a, $b, $c);
```

```
// This code is contributed by anuj_67.
?>
```

Output :

```
30
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/middle-of-three-using-minimum-comparisons/

Chapter 170

Minimize (max(A[i], B[j], C[k]) – min(A[i], B[j], C[k])) of three different sorted arrays

Minimize (max(A[i], B[j], C[k]) - min(A[i], B[j], C[k])) of three different sorted arrays - GeeksforGeeks

Given three sorted arrays A, B, and C of not necessarily same sizes. Calculate the minimum absolute difference between the maximum and minimum number of any triplet A[i], B[j], C[k] such that they belong to arrays A, B and C respectively, i.e., minimize (max(A[i], B[j], C[k]) – min(A[i], B[j], C[k]))

Examples:

```
Input : A : [ 1, 4, 5, 8, 10 ]
        B : [ 6, 9, 15 ]
        C : [ 2, 3, 6, 6 ]
Output : 1
Explanation: When we select A[i] = 5
B[j] = 6, C[k] = 6, we get the minimum difference
as max(A[i], B[j], C[k]) - min(A[i], B[j], C[k]))
= |6-5| = 1

Input : A = [ 5, 8, 10, 15 ]
        B = [ 6, 9, 15, 78, 89 ]
        C = [ 2, 3, 6, 6, 8, 8, 10 ]
Output : 1
Explanation: When we select A[i] = 10
b[j] = 9, C[k] = 10.
```

Start with the largest elements in each of the arrays A, B & C. Maintain a variable to

update the answer during each of the steps to be followed.
In every step, the only possible way to decrease the difference is to decrease the maximum element out of the three elements.
So traverse to the next largest element in the array containing the maximum element for this step and update the answer variable.
Repeat this step until the array containing the maximum element ends.

C++

```
 // C++ code for above approach
#include<bits/stdc++.h>
using namespace std;

int solve(int A[], int B[], int C[], int i, int j, int k)
{
        int min_diff, current_diff, max_term;

        // calculating min difference from last
        // index of lists
        min_diff = abs(max(A[i], max(B[j], C[k]))
                     - min(A[i], min(B[j], C[k])));

        while (i != -1 && j != -1 && k != -1)
        {
            current_diff = abs(max(A[i], max(B[j], C[k]))
                            - min(A[i], min(B[j], C[k])));

            // checking condition
            if (current_diff < min_diff)
                min_diff = current_diff;

            // calculating max term from list
            max_term = max(A[i], max(B[j], C[k]));

            // Moving to smaller value in the
            // array with maximum out of three.
            if (A[i] == max_term)
                i -= 1;
            else if (B[j] == max_term)
                j -= 1;
            else
                k -= 1;
        }

        return min_diff;
    }

    // Driver code
```

```
    int main()
    {
        int A[] = { 11, 4, 5, 8, 10 };
        int B[] = { 6, 9, 11 };
        int C[] = { 2, 3, 16, 6 };

        // Length of arrays.
        int nA = sizeof(A) / sizeof(A[0]);
        int nB = sizeof(B) / sizeof(B[0]);
        int nC = sizeof(C) / sizeof(C[0]);
        cout << solve(A, B, C, nA-1, nB-1, nC-1) << endl;

        int D[] = { 5, 8, 10, 15 };
        int E[] = { 6, 9, 15, 78, 89 };
        int F[] = { 2, 3, 6, 6, 8, 8, 10 };
        int nD = sizeof(D) / sizeof(D[0]);
        int nE = sizeof(E) / sizeof(E[0]);
        int nF = sizeof(F) / sizeof(F[0]);
        cout << solve(D, E, F, nD-1, nE-1, nF-1);

        return 0;
    }

// This code is contributed by Ravi Maurya.
```

Java

```
 // Java code for above approach
import java.io.*;

class GFG
{
    static int solve(int[] A, int[] B, int[] C)
    {
        int i, j, k;

        // assigning the length -1 value
        // to each of three variables
        i = A.length - 1;
        j = B.length - 1;
        k = C.length - 1;

        int min_diff, current_diff, max_term;

        // calculating min difference
        // from last index of lists
        min_diff = Math.abs(Math.max(A[i], Math.max(B[j], C[k]))
                  - Math.min(A[i], Math.min(B[j], C[k])));
```

```
        while (i != -1 && j != -1 && k != -1)
        {
            current_diff = Math.abs(Math.max(A[i], Math.max(B[j], C[k]))
                          - Math.min(A[i], Math.min(B[j], C[k])));

            // checking condition
            if (current_diff < min_diff)
                min_diff = current_diff;

            // calculating max term from list
            max_term = Math.max(A[i], Math.max(B[j], C[k]));

            // Moving to smaller value in the
            // array with maximum out of three.
            if (A[i] == max_term)
                i -= 1;
            else if (B[j] == max_term)
                j -= 1;
            else
                k -= 1;
        }
        return min_diff;
    }

    // Driver code
    public static void main(String[] args)
    {
        int[] A = { 11, 4, 5, 8, 10 };
        int[] B = { 6, 9, 11 };
        int[] C = { 2, 3, 16, 6 };
        System.out.println(solve(A, B, C));
        int[] D = { 5, 8, 10, 15 };
        int[] E = { 6, 9, 15, 78, 89 };
        int[] F = { 2, 3, 6, 6, 8, 8, 10 };
        System.out.println(solve(D, E, F));

    }
}
// This code is contributed by Gitanjali
```

Python 3

```
 # python code for above approach.

def solve(A, B, C):

        # assigning the length -1 value
```

```
        # to each of three variables
        i = len(A) - 1
        j = len(B) - 1
        k = len(C) - 1

        # calculating min difference
        # from last index of lists
        min_diff = abs(max(A[i], B[j], C[k]) -
        min(A[i], B[j], C[k]))

        while i != -1 and j != -1 and k != -1:
            current_diff = abs(max(A[i], B[j],
            C[k]) - min(A[i], B[j], C[k]))

            # checking condition
            if current_diff < min_diff:
                min_diff = current_diff

            # calculating max term from list
            max_term = max(A[i], B[j], C[k])

            # Moving to smaller value in the
            # array with maximum out of three.
            if A[i] == max_term:
                i -= 1
            elif B[j] == max_term:
                j -= 1
            else:
                k -= 1
        return min_diff

# driver code
A = [ 1, 4, 5, 8, 10 ]
B = [ 6, 9, 15 ]
C = [ 2, 3, 6, 6 ]
print(solve(A, B, C))
A = [ 5, 8, 10, 15 ]
B = [ 6, 9, 15, 78, 89 ]
C = [ 2, 3, 6, 6, 8, 8, 10 ]
print(solve(A, B, C))
```

C#

```
 // C# code for above approach
using System;

class GFG
{
```

```
static int solve(int[] A, int[] B, int[] C)
{
    int i, j, k;

    // assigning the length -1 value
    // to each of three variables
    i = A.Length - 1;
    j = B.Length - 1;
    k = C.Length - 1;

    int min_diff, current_diff, max_term;

    // calculating min difference
    // from last index of lists
    min_diff = Math.Abs(Math.Max(A[i], Math.Max(B[j], C[k]))
            - Math.Min(A[i], Math.Min(B[j], C[k])));

    while (i != -1 && j != -1 && k != -1)
    {
        current_diff = Math.Abs(Math.Max(A[i], Math.Max(B[j], C[k]))
                    - Math.Min(A[i], Math.Min(B[j], C[k])));

        // checking condition
        if (current_diff < min_diff)
            min_diff = current_diff;

        // calculating max term from list
        max_term = Math.Max(A[i], Math.Max(B[j], C[k]));

        // Moving to smaller value in the
        // array with maximum out of three.
        if (A[i] == max_term)
            i -= 1;
        else if (B[j] == max_term)
            j -= 1;
        else
            k -= 1;
    }
    return min_diff;
}

// Driver code
public static void Main()
{
    int[] A = { 11, 4, 5, 8, 10 };
    int[] B = { 6, 9, 11 };
    int[] C = { 2, 3, 16, 6 };
    Console.WriteLine(solve(A, B, C));
```

```
        int[] D = { 5, 8, 10, 15 };
        int[] E = { 6, 9, 15, 78, 89 };
        int[] F = { 2, 3, 6, 6, 8, 8, 10 };
        Console.WriteLine(solve(D, E, F));

    }
}
// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP code for above approach
function solve($A, $B, $C,
               $i, $j, $k)
{
    $min_diff;
    $current_diff;
    $max_term;

    // calculating min difference
    // from last index of lists
    $min_diff = abs(max($A[$i], max($B[$j], $C[$k])) -
                    min($A[$i], min($B[$j], $C[$k])));

    while ($i != -1 &&
           $j != -1 && $k != -1)
    {
        $current_diff = abs(max($A[$i], max($B[$j], $C[$k])) -
                            min($A[$i], min($B[$j], $C[$k])));

        // checking condition
        if ($current_diff < $min_diff)
            $min_diff = $current_diff;

        // calculating max term from list
        $max_term = max($A[$i],
                    max($B[$j], $C[$k]));

        // Moving to smaller value in the
        // array with maximum out of three.
        if ($A[$i] == $max_term)
            $i -= 1;
        else if ($B[$j] == $max_term)
            $j -= 1;
        else
            $k -= 1;
    }
```

```
    return $min_diff;
}

// Driver code
$A = array(11, 4, 5, 8, 10);
$B = array(6, 9, 11);
$C = array(2, 3, 16, 6);

// Length of arrays.
$nA = sizeof($A);
$nB = sizeof($B);
$nC = sizeof($C);
echo solve($A, $B, $C,
           $nA - 1, $nB - 1,
           $nC - 1) , "\n";

$D = array (5, 8, 10, 15);
$E = array (6, 9, 15, 78, 89);
$F = array (2, 3, 6, 6, 8, 8, 10);

$nD = sizeof($D) ;
$nE = sizeof($E) ;
$nF = sizeof($F);
echo solve($D, $E, $F,
           $nD - 1, $nE - 1,
           $nF - 1);

// This code is contributed by ajit
?>
```

Output:

```
1
1
```

Improved By : Ravi_Maurya, jit_t

Source

https://www.geeksforgeeks.org/minimize-maxai-bj-ck-minai-bj-ck-three-different-sorted-arrays/

Chapter 171

Minimize the sum of roots of a given polynomial

Minimize the sum of roots of a given polynomial - GeeksforGeeks

Given an array whose elements represents the coefficients of a polynomial of degree n, if the polynomial has a degree n then the array will have n+1 elements (one extra for the constant of a polynomial). Swap some elements of the array and print the resulting array such that the sum of the roots of the given polynomial is as minimum as possible irrespective of the nature of the roots.

Note that : except the first element of the array elements can be 0 also and the degree of the polynomial is always greater than 1.

Examples:

```
Input : -4 1 6 -3 -2 -1
Output : 1 6 -4 -3 -2 -1
        Here, the array is -4, 1, 6, -3, -2, -1
        i.e the polynomial is -4.x^5 + 1.x^4 + 6.x^3 - 3.x^2 - 2.x^1 - 1
        minimum sum = -6

Input : -9 0 9
Output :-9 0 9
       Here polynomial is
       -9.x^2 + 0.x^1 + 9
       minimum sum = 0
```

Solution : Let us recall the fact about the sum of the roots of a polynomial if a polynomial p(x) = a.x^n + b.x^n-1 + c.x^n-2 + … + k, then the sum of roots of a polynomial is given by **-b/a**. Please see Vieta's formulas for details.

We have to minimize -b/a i.e to maximize b/a i.e maximize b and minimize a. So if somehow we are able to maximize b and minimize a, we will swap the values of the coefficients and copy rest of the array as it is.

There will be four cases :
Case #1: when the number of positive coefficients and the number of negative coefficients both are greater than or equal to 2
In this case, we will find a maximum and minimum from positive elements and from negative elements also and we will check -(maxPos)/(minPos) is smaller or -(abs(maxNeg))/ (abs(minNeg)) is smaller and print the answer after swapping accordingly.

Case #2: when the number of positive coefficients is greater than equal to 2 but the number of negative coefficients is less than 2
In this case, we will consider the case of the maximum of positive and minimum of positive elements only. Because if we picked up one from positive elements and the other from negative elements the result of -b/a will be a positive value which is not minimum. (as we require a large negative value)

Case #3: when the number of negative coefficients is greater than equal to 2 but the number of positive coefficients is less than 2
In this case, we will consider the case of the maximum of negative and minimum of negative elements only. Because if we picked up one from positive elements and the other from negative elements the result of -b/a will be a positive value which is not minimum. (as we require a large negative value)

Case #4: When both the counts are less than or equal to 1
Observe carefully, You cannot swap elements in this case.

```
 // CPP program to find minimum sum of roots of a
// given polynomial
#include <bits/stdc++.h>
using namespace std;

void getMinimumSum(int arr[], int n)
{
    // resultant vector
    vector<int> res;

    // a vector that store indices of the positive
    // elements
    vector<int> pos;

    // a vector that store indices of the negative
    // elements
    vector<int> neg;

    for (int i = 0; i < n; i++) {
        if (arr[i] > 0)
            pos.push_back(i);
```

```
        else if (arr[i] < 0)
            neg.push_back(i);
    }

    // Case - 1:
    if (pos.size() >= 2 && neg.size() >= 2) {
        int posMax = INT_MIN, posMaxIdx = -1;
        int posMin = INT_MAX, posMinIdx = -1;

        int negMax = INT_MIN, negMaxIdx = -1;
        int negMin = INT_MAX, negMinIdx = -1;

        for (int i = 0; i < pos.size(); i++) {
            if (arr[pos[i]] > posMax) {
                posMaxIdx = pos[i];
                posMax = arr[posMaxIdx];
            }
        }

        for (int i = 0; i < pos.size(); i++) {
            if (arr[pos[i]] < posMin &&
                pos[i] != posMaxIdx) {
                posMinIdx = pos[i];
                posMin = arr[posMinIdx];
            }
        }

        for (int i = 0; i < neg.size(); i++) {
            if (abs(arr[neg[i]]) > negMax) {
                negMaxIdx = neg[i];
                negMax = abs(arr[negMaxIdx]);
            }
        }

        for (int i = 0; i < neg.size(); i++) {
            if (abs(arr[neg[i]]) < negMin &&
                neg[i] != negMaxIdx) {
                negMinIdx = neg[i];
                negMin = abs(arr[negMinIdx]);
            }
        }

        double posVal = -1.0 * (double)posMax / (double)posMin;
        double negVal = -1.0 * (double)negMax / (double)negMin;

        if (posVal < negVal) {
            res.push_back(arr[posMinIdx]);
            res.push_back(arr[posMaxIdx]);
```

```
        for (int i = 0; i < n; i++) {
            if (i != posMinIdx && i != posMaxIdx) {
                res.push_back(arr[i]);
            }
        }
    }
    else {
        res.push_back(arr[negMinIdx]);
        res.push_back(arr[negMaxIdx]);

        for (int i = 0; i < n; i++) {
            if (i != negMinIdx && i != negMaxIdx) {
                res.push_back(arr[i]);
            }
        }
    }
    for (int i = 0; i < res.size(); i++) {
        cout << res[i] << " ";
    }
    cout << "\n";
}

// Case - 2:
else if (pos.size() >= 2) {
    int posMax = INT_MIN, posMaxIdx = -1;
    int posMin = INT_MAX, posMinIdx = -1;

    for (int i = 0; i < pos.size(); i++) {
        if (arr[pos[i]] > posMax) {
            posMaxIdx = pos[i];
            posMax = arr[posMaxIdx];
        }
    }

    for (int i = 0; i < pos.size(); i++) {
        if (arr[pos[i]] < posMin &&
            pos[i] != posMaxIdx) {
            posMinIdx = pos[i];
            posMin = arr[posMinIdx];
        }
    }

    res.push_back(arr[posMinIdx]);
    res.push_back(arr[posMaxIdx]);

    for (int i = 0; i < n; i++) {
        if (i != posMinIdx && i != posMaxIdx) {
```

```
                res.push_back(arr[i]);
            }
        }
        for (int i = 0; i < res.size(); i++) {
            cout << res[i] << " ";
        }
        cout << "\n";
    }

    // Case - 3:
    else if (neg.size() >= 2) {
        int negMax = INT_MIN, negMaxIdx = -1;
        int negMin = INT_MAX, negMinIdx = -1;

        for (int i = 0; i < neg.size(); i++) {
            if (abs(arr[neg[i]]) > negMax) {
                negMaxIdx = neg[i];
                negMax = abs(arr[negMaxIdx]);
            }
        }

        for (int i = 0; i < neg.size(); i++) {
            if (abs(arr[neg[i]]) < negMin &&
                neg[i] != negMaxIdx) {
                negMinIdx = neg[i];
                negMin = abs(arr[negMinIdx]);
            }
        }

        res.push_back(arr[negMinIdx]);
        res.push_back(arr[negMaxIdx]);

        for (int i = 0; i < n; i++)
            if (i != negMinIdx && i != negMaxIdx)
                res.push_back(arr[i]);

        for (int i = 0; i < res.size(); i++)
            cout << res[i] << " ";

        cout << "\n";
    }

    // Case - 4:
    else {
        cout << "No swap required\n";
    }
}
```

```
int main()
{
    int arr[] = { -4, 1, 6, -3, -2, -1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    getMinimumSum(arr, n);
    return 0;
}
```

Output:

```
1 6 -4 -3 -2 -1
```

Source

https://www.geeksforgeeks.org/minimize-sum-roots-given-polynomial/

Chapter 172

Minimum De-arrangements present in array of AP (Arithmetic Progression)

Minimum De-arrangements present in array of AP (Arithmetic Progression) - GeeksforGeeks

Given an array of n-elements. Given array is a permutation of some Arithmetic Progression. Find the minimum number of De-arrangements present in that array so as to make that array an Arithmetic progression.

Examples:

```
Input : arr[] = [8, 6, 10 ,4, 2]
Output : Minimum De-arrangement = 3
Explanation : arr[] = [10, 8, 6, 4, 2] is permutation
which forms an AP and has minimum de-arrangements.

Input : arr[] = [5, 10, 15, 25, 20]
Output : Minimum De-arrangement = 2
Explanation : arr[] = [5, 10, 15, 20, 25] is permutation
which forms an AP and has minimum de-arrangements.
```

As per property of Arithmetic Progression our sequence will be either in increasing or decreasing manner. Also, we know that reverse of any Arithmetic Progression also form another Arithmetic Progression. So, we create a copy of original array and then once sort our given array in increase order and find total count of mismatch again after that we will reverse our sorted array and found new count of mismatch. Comparing both the counts of mismatch we can find the minimum number of de-arrangements. Time Complexity = O(nlogn).

C++

```
 // CPP for counting minimum de-arrangements present
// in an array.
#include<bits/stdc++.h>
using namespace std;

// function to count Dearrangement
int countDe (int arr[], int n)
{
    // create a copy of original array
    vector <int> v (arr, arr+n);

    // sort the array
    sort(arr, arr+n);

    // traverse sorted array for counting mismatches
    int count1 = 0;
    for (int i=0; i<n; i++)
        if (arr[i] != v[i])
            count1++;

    // reverse the sorted array
    reverse(arr,arr+n);

    // traverse reverse sorted array for counting
    // mismatches
    int count2 = 0;
    for (int i=0; i<n; i++)
        if (arr[i] != v[i])
            count2++;

    // return minimum mismatch count
    return (min (count1, count2));
}

// driver program
int main()
{
    int arr[] = {5, 9, 21, 17, 13};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Minimum Dearrangement = " << countDe(arr, n);
    return 0;
}
```

Java

```
 // Java code for counting minimum
// de-arrangements present in an array.
import java.util.*;
```

```
import java.lang.*;
import java.util.Arrays;

public class GeeksforGeeks{

    // function to count Dearrangement
    public static int countDe(int arr[], int n){
        int v[] = new int[n];

        // create a copy of original array
        for(int i = 0; i < n; i++)
            v[i] = arr[i];

        // sort the array
        Arrays.sort(arr);

        // traverse sorted array for
        // counting mismatches
        int count1 = 0;
        for (int i = 0; i < n; i++)
            if (arr[i] != v[i])
            count1++;

        // reverse the sorted array
        Collections.reverse(Arrays.asList(arr));

        // traverse reverse sorted array
        // for counting mismatches
        int count2 = 0;
        for (int i = 0; i < n; i++)
            if (arr[i] != v[i])
                count2++;

        // return minimum mismatch count
        return (Math.min (count1, count2));
    }

    // driver code
    public static void main(String argc[]){
        int arr[] = {5, 9, 21, 17, 13};
        int n = 5;
        System.out.println("Minimum Dearrangement = "+
                           countDe(arr, n));
    }
}

/*This code is contributed by Sagar Shukla.*/
```

Python3

```
 # Python3 code for counting minimum
# de-arrangements present in an array.

# function to count Dearrangement
def countDe(arr, n):
        v = [None] * n

        i=0

        # create a copy of
        # original array
        while(i < n):
            v[i] = arr[i]
            i = i + 1

        # sort the array
        arr.sort()

        # traverse sorted array for
        # counting mismatches
        count1 = 0
        i = 0
        while( i < n ):
            if (arr[i] != v[i]):
                count1 = count1 + 1
            i = i + 1

        # reverse the sorted array
        arr.sort(reverse=True)

        # traverse reverse sorted array
        # for counting mismatches
        count2 = 0
        i = 0
        while( i < n ):
            if (arr[i] != v[i]):
                count2 = count2 + 1
            i = i + 1

        # return minimum mismatch count
        return (min (count1, count2))

# Driven code
arr = [5, 9, 21, 17, 13]
n = 5
print ("Minimum Dearrangement =",countDe(arr, n))
```

```
# This code is contributed by "rishabh_jain".
```

C#

```
 // C# code for counting
// minimum de-arrangements
// present in an array.
using System;

class GFG
{

// function to count
// Dearrangement
public static int countDe(int[] arr,
                          int n)
{
    int[] v = new int[n];

    // create a copy
    // of original array
    for(int i = 0; i < n; i++)
        v[i] = arr[i];

    // sort the array
    Array.Sort(arr);

    // traverse sorted array for
    // counting mismatches
    int count1 = 0;
    for (int i = 0; i < n; i++)
        if (arr[i] != v[i])
        count1++;

    // reverse the sorted array
    Array.Reverse(arr);

    // traverse reverse sorted array
    // for counting mismatches
    int count2 = 0;
    for (int i = 0; i < n; i++)
        if (arr[i] != v[i])
            count2++;

    // return minimum
    // mismatch count
    return (Math.Min (count1, count2));
```

```
}

// Driver code
public static void Main()
{
    int[] arr = new int[]{5, 9, 21, 17, 13};
    int n = 5;
    Console.WriteLine("Minimum Dearrangement = " +
                                countDe(arr, n));
}
}

// This code is contributed by mits
```

Output:

```
Minimum Dearrangement = 2
```

Improved By : Mithun Kumar

Source

https://www.geeksforgeeks.org/minimum-de-arrangements-present-array-ap-arithmetic-progression/

Chapter 173

Minimum absolute difference of adjacent elements in a circular array

Minimum absolute difference of adjacent elements in a circular array - GeeksforGeeks

Given n integers, which form a circle. Find the minimal absolute value of any adjacent pair. If there are many optimum solutions, output any of them.
Note: they are in circle

Examples:

```
Input : arr[] = {10, 12, 13, 15, 10}
Output : 0
Explanation: |10 - 10| = 0 which is the
minimum possible.

Input : arr[] = {10, 20, 30, 40}
Output : 10
Explanation: |10 - 20| = 10 which is the
minimum, 2 3 or 3 4 can be the answers also.
```

Initially consider the minimum value to be of first and second elements. Traverse from second element to last. Check the difference of every adjacent pair and store the minimum value. When last element is reached, check its difference with first element.

Below is the implementation of the above approach.

C++

```
// C++ program to find maximum difference
```

```
// between adjacent elements in a circular array.
#include <bits/stdc++.h>
using namespace std;

void minAdjDifference(int arr[], int n)
{
    if (n < 2)
        return;

    // Checking normal adjacent elements
    int res = abs(arr[1] - arr[0]);
    for (int i = 2; i < n; i++)
        res = min(res, abs(arr[i] - arr[i - 1]));

    // Checking circular link
    res = min(res, abs(arr[n - 1] - arr[0]));

    cout << "Min Difference = " << res;
}

// driver program to check the above function
int main()
{
    int a[] = { 10, 12, 13, 15, 10 };
    int n = sizeof(a) / sizeof(a[0]);
    minAdjDifference(a, n);
    return 0;
}
```

Java

```
 // Java program to find maximum difference
// between adjacent elements in a circular
// array.
class GFG {

    static void minAdjDifference(int arr[], int n)
    {
        if (n < 2)
            return;

        // Checking normal adjacent elements
        int res = Math.abs(arr[1] - arr[0]);
        for (int i = 2; i < n; i++)
            res = Math.min(res, Math.abs(arr[i] - arr[i - 1]));

        // Checking circular link
        res = Math.min(res, Math.abs(arr[n - 1] - arr[0]));
```

```
        System.out.print("Min Difference = " + res);
    }

    // driver code
    public static void main(String arg[])
    {

        int a[] = { 10, 12, 13, 15, 10 };
        int n = a.length;

        minAdjDifference(a, n);
    }
}

// This code is contributed by Anant Agarwal
// and improved by Anuj Sharma.
```

Python3

```
# Python3 program to find maximum
# difference between adjacent
# elements in a circular array.

def minAdjDifference(arr, n):

    if (n < 2): return

    # Checking normal adjacent elements
    res = abs(arr[1] - arr[0])

    for i in range(2, n):
        res = min(res, abs(arr[i] - arr[i - 1]))

    # Checking circular link
    res = min(res, abs(arr[n - 1] - arr[0]))

    print("Min Difference = ", res)

# Driver Code
a = [10, 12, 13, 15, 10]
n = len(a)
minAdjDifference(a, n)

# This code is contributed by Anant Agarwal
# and improved by Anuj Sharma.
```

C#

```
 // C# program to find maximum difference
// between adjacent elements in a circular array.
using System;

class GFG {

    static void minAdjDifference(int[] arr, int n)
    {
        if (n < 2)
            return;

        // Checking normal adjacent elements
        int res = Math.Abs(arr[1] - arr[0]);
        for (int i = 2; i < n; i++)
            res = Math.Min(res, Math.Abs(arr[i] - arr[i - 1]));

        // Checking circular link
        res = Math.Min(res, Math.Abs(arr[n - 1] - arr[0]));

        Console.Write("Min Difference = " + res);
    }

    // driver code
    public static void Main()
    {
        int[] a = { 10, 12, 13, 15, 10 };
        int n = a.Length;
        minAdjDifference(a, n);
    }
}

// This code is contributed by Anant Agarwal
// and improved by Anuj Sharma.
```

PHP

```
 <?php
// PHP program to find maximum
// difference between adjacent
// elements in a circular array.

function minAdjDifference($arr, $n)
{
    if ($n < 2)
        return;
```

```
    // Checking normal
    // adjacent elements
    $res = abs($arr[1] - $arr[0]);
    for ($i = 2; $i < $n; $i++)
        $res = min($res,
               abs($arr[$i] -
               $arr[$i - 1]));

    // Checking circular link
    $res = min($res, abs($arr[$n - 1] -
                    $arr[0]));

    echo "Min Difference = ", $res;
}

// Driver Code
$a = array(10, 12, 13, 15, 10);
$n = count($a);
minAdjDifference($a, $n);

//This code is contributed by anuj_67
//and improved by Anuj Sharma.
?>
```

Output:

```
Min Difference =  0
```

Time complexity: O(n)

Improved By : vt_m, Anuj_Sharma

Source

https://www.geeksforgeeks.org/minimum-absolute-difference-adjacent-elements-circular-array/

Chapter 174

Minimum distance between two occurrences of maximum

Minimum distance between two occurrences of maximum - GeeksforGeeks

You are given an array of n-elements with a basic condition that occurrence of greatest element is more than once. You have to find the minimum distance between maximums. (n>=2).

Examples:

```
Input : arr[] = {3, 5, 2, 3, 5, 3, 5}
Output : Minimum Distance = 2
Explanation : Greatest element is 5 and its index
are 1, 4 and 6. Resulting minimum distance of 2
from position 4 to 6.

Input : arr[] = {1, 1, 1, 1, 1, 1}
Output : Minimum Distance = 1
Explanation : Greatest element is 1 and its index
are 0, 1, 2, 3, 4 and 5. Resulting minimum distance
of 1.
```

A **basic approach** runs in $O(n^2)$. First we find the maximum element. Then for each element equal to maximum element, we find the nearest maximum element.
An **efficient solution** finish our job in a single traversing of array. We initialize maximum_element = arr[0], min_distance = n and index = 0. After that for each element we should check whether element is equal to, greater or less than maximum element. Depending upon three cases we have following option:

- **case a:** If element is equal to maximum_element then update min_dis = min(min_dis , (i-index)) and update index = i;

- **case b:** If element is greater than maximum_element update maximum_element = arr[i], index = i and min_dis = n.
- **case c:** If element is less than maximum_element then iterate to next element.

C

```
 // C program to find Min distance
// of maximum element
#include<bits/stdc++.h>
using namespace std;

//function to return min distance
int minDistance (int arr[], int n)
{
    int maximum_element = arr[0];
    int min_dis = n;
    int index = 0;

    for (int i=1; i<n; i++)
    {
        // case a
        if (maximum_element == arr[i])
        {
            min_dis = min(min_dis, (i-index));
            index = i;
        }

        // case b
        else if (maximum_element < arr[i])
        {
            maximum_element = arr[i];
            min_dis = n;
            index = i;
        }

        // case c
        else
            continue;
    }

    return min_dis;
}

// driver program
int main()
{
    int arr[] = {6, 3, 1, 3, 6, 4, 6};
    int n = sizeof(arr)/sizeof(arr[0]);
```

```
    cout << "Minimum distance = "
         << minDistance(arr, n);
    return 0;
}
```

Java

```
 // Java program to find Min distance
// of maximum element
class GFG
{

    // function to return min distance
    static int minDistance (int arr[], int n)
    {
        int maximum_element = arr[0];
        int min_dis = n;
        int index = 0;

        for (int i=1; i<n; i++)
        {

            // case a
            if (maximum_element == arr[i])
            {
                min_dis = Math.min(min_dis, (i-index));
                index = i;
            }

            // case b
            else if (maximum_element < arr[i])
            {
                maximum_element = arr[i];
                min_dis = n;
                index = i;
            }

            // case c
            else
                continue;
        }

        return min_dis;
    }

    // Driver code
    public static void main (String[] args)
    {
```

```
        int arr[] = {6, 3, 1, 3, 6, 4, 6};
        int n = arr.length;
        System.out.print("Minimum distance = "+minDistance(arr, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find Min
# distance of maximum element

# Function to return min distance
def minDistance (arr, n):

    maximum_element = arr[0]
    min_dis = n
    index = 0

    for i in range(1, n):

        # case a
        if (maximum_element == arr[i]):

            min_dis = min(min_dis, (i - index))
            index = i


        # case b
        elif (maximum_element < arr[i]):

            maximum_element = arr[i]
            min_dis = n
            index = i


        # case c
        else:
            continue


    return min_dis


# driver program
arr = [6, 3, 1, 3, 6, 4, 6]
n = len(arr)
```

```
print("Minimum distance =", minDistance(arr, n))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to find Min distance
// of maximum element
using System;

class GFG {

    // function to return min distance
    static int minDistance (int []arr, int n)
    {
        int maximum_element = arr[0];
        int min_dis = n;
        int index = 0;

        for (int i = 1; i < n; i++)
        {

            // case a
            if (maximum_element == arr[i])
            {
                min_dis = Math.Min(min_dis,
                                (i - index));
                index = i;
            }

            // case b
            else if (maximum_element < arr[i])
            {
                maximum_element = arr[i];
                min_dis = n;
                index = i;
            }

            // case c
            else
                continue;
        }

        return min_dis;
    }

    // Driver code
    public static void Main ()
```

```
    {
        int []arr = {6, 3, 1, 3, 6, 4, 6};
        int n = arr.Length;

        Console.WriteLine("Minimum distance = "
                        + minDistance(arr, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find Min distance
// of maximum element

//function to return min distance
function minDistance($arr, $n)
{
    $maximum_element = $arr[0];
    $min_dis = $n;
    $index = 0;

    for ($i = 1; $i < $n; $i++)
    {

        // case a
        if ($maximum_element == $arr[$i])
        {
            $min_dis = min($min_dis, ($i - $index));
            $index = $i;
        }

        // case b
        else if ($maximum_element < $arr[$i])
        {
            $maximum_element = $arr[$i];
            $min_dis = $n;
            $index = $i;
        }

        // case c
        else
            continue;
    }

return $min_dis;
```

```
}

    // Driver Code
    $arr = array(6, 3, 1, 3, 6, 4, 6);
    $n = count($arr);
    echo "Minimum distance = "
          .minDistance($arr,$n);

// This code is contributed by Sam007
?>
```

Output :

```
Minimum distance = 2
```

Improved By : Sam007

Source

https://www.geeksforgeeks.org/minimum-distance-between-two-occurrences-of-maximum/

Chapter 175

Minimum increment by k operations to make all elements equal

Minimum increment by k operations to make all elements equal - GeeksforGeeks

You are given an array of n-elements, you have to find the number of operations needed to make all elements of array equal. Where a single operation can increment an element by k. If it is not possible to make all elements equal print -1.

Example :

```
Input : arr[] = {4, 7, 19, 16},  k = 3
Output : 8

Input : arr[] = {4, 4, 4, 4}, k = 3
Output : 0

Input : arr[] = {4, 2, 6, 8}, k = 3
Output : -1
```

To solve this question we require to check whether all elements can became equal or not and that too only by incrementing k from elements value. For this we have to check that the difference of any two elements should always be divisible by k. If it is so, then all elements can become equal otherwise they can not became equal in any case by incrementing k from them. Also, the number of operations required can be calculated by finding value of (max – Ai)/k for all elements. where max is maximum element of array.
Algorithm :

```
// iterate for all elements
for (int i=0; i<n; i++)
{
    // check if element can make equal to max
    //  or not if not then return -1
    if ((max - arr[i]) % k != 0 )
        return -1;

    // else update res for required operations
    else
        res += (max - arr[i]) / k ;
}

return res;
```

C++

```
 // Program to make all array equal
#include <bits/stdc++.h>
using namespace std;

// function for calculating min operations
int minOps(int arr[], int n, int k)
{
    // max elements of array
    int max = *max_element(arr, arr + n);
    int res = 0;

    // iterate for all elements
    for (int i = 0; i < n; i++) {

        // check if element can make equal to
        // max or not if not then return -1
        if ((max - arr[i]) % k != 0)
            return -1;

        // else update res for required operations
        else
            res += (max - arr[i]) / k;
    }

    // return result
    return res;
}

// driver program
int main()
{
```

```
    int arr[] = { 21, 33, 9, 45, 63 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 6;
    cout << minOps(arr, n, k);
    return 0;
}
```

Java

```
 // Program to make all array equal
import java.io.*;
import java.util.Arrays;

class GFG {
    // function for calculating min operations
    static int minOps(int arr[], int n, int k)
    {
        // max elements of array
        Arrays.sort(arr);
        int max = arr[arr.length - 1];
        int res = 0;

        // iterate for all elements
        for (int i = 0; i < n; i++) {

            // check if element can make equal to
            // max or not if not then return -1
            if ((max - arr[i]) % k != 0)
                return -1;

            // else update res for required operations
            else
                res += (max - arr[i]) / k;
        }

        // return result
        return res;
    }

    // Driver program
    public static void main(String[] args)
    {
        int arr[] = { 21, 33, 9, 45, 63 };
        int n = arr.length;
        int k = 6;
        System.out.println(minOps(arr, n, k));
    }
}
```

```
// This code is contributed by vt_m
```

Python3

```
 # Python3 Program to make all array equal

# function for calculating min operations
def minOps(arr, n, k):

    # max elements of array
    max1 = max(arr)
    res = 0

    # iterate for all elements
    for i in range(0, n):

        # check if element can make equal to
        # max or not if not then return -1
        if ((max1 - arr[i]) % k != 0):
            return -1

        # else update res fo
        # required operations
        else:
            res += (max1 - arr[i]) / k

    # return result
    return int(res)

# driver program
arr = [21, 33, 9, 45, 63]
n = len(arr)
k = 6
print(minOps(arr, n, k))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program Minimum increment by
// k operations to make all elements equal.
using System;

class GFG {
```

```
    // function for calculating min operations
    static int minOps(int[] arr, int n, int k)
    {

        // max elements of array
        Array.Sort(arr);
        int max = arr[arr.Length - 1];
        int res = 0;

        // iterate for all elements
        for (int i = 0; i < n; i++) {

            // check if element can make
            // equal to max or not if not
            // then return -1
            if ((max - arr[i]) % k != 0)
                return -1;

            // else update res for required
            // operations
            else
                res += (max - arr[i]) / k;
        }

        // return result
        return res;
    }

    // Driver program
    public static void Main()
    {
        int[] arr = { 21, 33, 9, 45, 63 };
        int n = arr.Length;
        int k = 6;

        Console.Write(minOps(arr, n, k));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// Program to make all array equal

// function for calculating
// min operations
```

```
function minOps($arr, $n, $k)
{
    // max elements of array
    $max = max($arr);
    $res = 0;

    // iterate for all elements
    for ($i = 0; $i < $n; $i++)
    {

        // check if element can
        // make equal to max or
        // not if not then return -1
        if (($max - $arr[$i]) % $k != 0)
            return -1;

        // else update res for
        // required operations
        else
            $res += ($max -
                     $arr[$i]) / $k;
    }

    // return result
    return $res;
}

// Driver Code
$arr = array(21, 33, 9, 45, 63);
$n = count($arr);
$k = 6;
print (minOps($arr, $n, $k));

// This code is contributed
// by Manish Shaw(manishshaw1)
?>
```

Output :

```
24
```

Improved By : nitin mittal, manishshaw1

Source

https://www.geeksforgeeks.org/minimum-increment-k-operations-make-elements-equal/

Chapter 176

Minimum number of points to be removed to get remaining points on one side of axis

Minimum number of points to be removed to get remaining points on one side of axis - GeeksforGeeks

We are given **n** points in a Cartesian plane. Our task is to find the minimum number of points that should be removed in order to get the remaining points on one side of any axis.

Examples :

```
Input : 4
        1 1
        2 2
       -1 -1
       -2 2
Output : 1
Explanation :
If we remove (-1, -1) then all the remaining
points are above x-axis. Thus the answer is 1.

Input : 3
        1 10
        2 3
        4 11
Output : 0
Explanation :
All points are already above X-axis. Hence the
answer is 0.
```

Approach :
This problem is a simple example of constructive brute force algorithm on Geometry. The solution can be approached simply by finding the number of points on all sides of the X-axis and Y-axis. The minimum of this will be the answer.

```
 // CPP program to find minimum points to be moved
// so that all points are on same side.
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

// Structure to store the coordinates of a point.
struct Point
{
    int x, y;
};

// Function to find the minimum number of points
int findmin(Point p[], int n)
{
    int a = 0, b = 0, c = 0, d = 0;
    for (int i = 0; i < n; i++)
    {
        // Number of points on the left of Y-axis.
        if (p[i].x <= 0)
            a++;

        // Number of points on the right of Y-axis.
        else if (p[i].x >= 0)
            b++;

        // Number of points above X-axis.
        if (p[i].y >= 0)
            c++;

        // Number of points below X-axis.
        else if (p[i].y <= 0)
            d++;
    }

    return min({a, b, c, d});
}

// Driver Function
int main()
{
    Point p[] = { {1, 1}, {2, 2}, {-1, -1}, {-2, 2} };
    int n = sizeof(p)/sizeof(p[0]);
```

```
    cout << findmin(p, n);
    return 0;
}
```

Output :

```
1
```

Source

https://www.geeksforgeeks.org/minimum-number-points-removed-get-remaining-points-one-side-axis/

Chapter 177

Minimum product pair an array of positive Integers

Minimum product pair an array of positive Integers - GeeksforGeeks

Given an array of positive integers. We are required to write a program to print the minimum product of any two numbers of the given array.

Examples:

```
Input : 11 8 5 7 5 100
Output : 25
Explanation : The minimum product of any
two numbers will be 5 * 5 = 25.

Input : 198 76 544 123 154 675
Output : 7448
Explanation : The minimum product of any
two numbers will be 76 * 123 = 7448.
```

Simple Approach : A simple approach will be to run two nested loops to generate all possible pair of elements and keep track of the minimum product.
Time Complexity: O(n * n)
Auxiliary Space: O(1)

Better Approach: An efficient approach will be to first sort the given array and print the product of first two numbers, sorting will take O(n log n). Answer will be then a[0] * a[1]
Time Complexity: O(n * log(n))
Auxiliary Space: O(1)

Best Approach: The idea is linearly traverse given array and keep track of minimum two elements. Finally return product of two minimum elements.

Below is the implementation of above approach.

C++

```
 // C++ program to calculate minimum
// product of a pair
#include <bits/stdc++.h>
using namespace std;

// Function to calculate minimum product
// of pair
int printMinimumProduct(int arr[], int n)
{
    // Initialize first and second
    // minimums. It is assumed that the
    // array has at least two elements.
    int first_min = min(arr[0], arr[1]);
    int second_min = max(arr[0], arr[1]);

    // Traverse remaining array and keep
    // track of two minimum elements (Note
    // that the two minimum elements may
    // be same if minimum element appears
    // more than once)
    // more than once)
    for (int i=2; i<n; i++)
    {
       if (arr[i] < first_min)
       {
          second_min = first_min;
          first_min = arr[i];
       }
       else if (arr[i] < second_min)
          second_min = arr[i];
    }

    return first_min * second_min;
}

// Driver program to test above function
int main()
{
    int a[] = { 11, 8 , 5 , 7 , 5 , 100 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << printMinimumProduct(a,n);
    return 0;
}
```

Java

```
 // Java program to calculate minimum
// product of a pair
import java.util.*;

class GFG {

    // Function to calculate minimum product
    // of pair
    static int printMinimumProduct(int arr[], int n)
    {
        // Initialize first and second
        // minimums. It is assumed that the
        // array has at least two elements.
        int first_min = Math.min(arr[0], arr[1]);
        int second_min = Math.max(arr[0], arr[1]);

        // Traverse remaining array and keep
        // track of two minimum elements (Note
        // that the two minimum elements may
        // be same if minimum element appears
        // more than once)
        // more than once)
        for (int i = 2; i < n; i++)
        {
           if (arr[i] < first_min)
           {
              second_min = first_min;
              first_min = arr[i];
           }
           else if (arr[i] < second_min)
              second_min = arr[i];
        }

        return first_min * second_min;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int a[] = { 11, 8 , 5 , 7 , 5 , 100 };
        int n = a.length;
        System.out.print(printMinimumProduct(a,n));

    }
}
```

```
// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python program to
# calculate minimum
# product of a pair

# Function to calculate
# minimum product
# of pair
def printMinimumProduct(arr,n):

    # Initialize first and second
    # minimums. It is assumed that the
    # array has at least two elements.
    first_min = min(arr[0], arr[1])
    second_min = max(arr[0], arr[1])

    # Traverse remaining array and keep
    # track of two minimum elements (Note
    # that the two minimum elements may
    # be same if minimum element appears
    # more than once)
    # more than once)
    for i in range(2,n):

         if (arr[i] < first_min):

            second_min = first_min
            first_min = arr[i]

         elif (arr[i] < second_min):
            second_min = arr[i]

    return first_min * second_min

# Driver code

a= [ 11, 8 , 5 , 7 , 5 , 100 ]
n = len(a)

print(printMinimumProduct(a,n))

# This code is contributed
# by Anant Agarwal.
```

C#

```
 // C# program to calculate minimum
// product of a pair
using System;

class GFG {

    // Function to calculate minimum
    // product of pair
    static int printMinimumProduct(int []arr,
                                   int n)
    {

        // Initialize first and second
        // minimums. It is assumed that
        // the array has at least two
        // elements.
        int first_min = Math.Min(arr[0],
                                 arr[1]);

        int second_min = Math.Max(arr[0],
                                 arr[1]);

        // Traverse remaining array and
        // keep track of two minimum
        // elements (Note that the two
        // minimum elements may be same
        // if minimum element appears
        // more than once)
        for (int i = 2; i < n; i++)
        {
            if (arr[i] < first_min)
            {
                second_min = first_min;
                first_min = arr[i];
            }
            else if (arr[i] < second_min)
                second_min = arr[i];
        }

        return first_min * second_min;
    }

    /* Driver program to test above
    function */
    public static void Main()
    {
        int []a = { 11, 8 , 5 , 7 ,
                            5 , 100 };
```

```
        int n = a.Length;

        Console.WriteLine(
            printMinimumProduct(a, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to calculate minimum
// product of a pair

// Function to calculate minimum
// product of pair
function printMinimumProduct($arr, $n)
{

    // Initialize first and second
    // minimums. It is assumed that the
    // array has at least two elements.
    $first_min = min($arr[0], $arr[1]);
    $second_min = max($arr[0], $arr[1]);

    // Traverse remaining array and keep
    // track of two minimum elements (Note
    // that the two minimum elements may
    // be same if minimum element appears
    // more than once)
    // more than once)
    for ($i = 2; $i < $n; $i++)
    {
        if ($arr[$i] < $first_min)
        {
            $second_min = $first_min;
            $first_min = $arr[$i];
        }
        else if ($arr[$i] < $second_min)
            $second_min = $arr[$i];
    }

    return $first_min * $second_min;
}

// Driver Code
$a = array(11, 8 , 5 , 7 , 5 , 100);
```

```
$n = sizeof($a);
echo(printMinimumProduct($a, $n));

// This code is contributed by Ajit.
?>
```

Output:

```
25
```

Time Complexity: O(n)
Auxiliary Space: O(1)

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/minimum-product-pair-an-array-of-positive-integers/

Chapter 178

Minimum time required to produce m items

Minimum time required to produce m items - GeeksforGeeks

Consider **n** machines which produce same type of items but at different rate i.e., machine 1 takes a_1 sec to produce an item, machine 2 takes a_2 sec to produce an item. Given an array which contains the time required by i^{th} machine to produce an item. Considering all machine are working simultaneously, the task is to find minimum time required to produce **m** items.

Examples:

```
Input : arr[] = {1, 2, 3}, m = 11
Output : 6
In 6 sec, machine 1 produces 6 items, machine 2 produces 3 items,
and machine 3 produces 2 items. So to produce 11 items minimum
6 sec are required.

Input : arr[] = {5, 6}, m = 11
Output : 30
```

Method 1 : (Brute Force)
Initialize time = 0 and increment it by 1. Calculate number of item produce at each time until number of produced items is not equal to m.
Below is the implementation of the above idea :

C++

```
 // C++ program to find minimum time
// required to produce m items.
#include<bits/stdc++.h>
```

```
using namespace std;

// Return the minimum time required to
// produce m items with given machines.
int minTime(int arr[], int n, int m)
{
    // Intialise time, items equal to 0.
    int t = 0;

    while (1)
    {
        int items = 0;

        // Calculating items at each second
        for (int i = 0; i < n; i++)
            items += (t / arr[i]);

        // If items equal to m return time.
        if (items >= m)
            return t;

        t++; // Increment time
    }
}

// Driver Code
int main()
{
    int arr[] = { 1, 2, 3 };
    int n = sizeof(arr)/sizeof(arr[0]);
    int m = 11;

    cout << minTime(arr, n, m) << endl;

    return 0;
}
```

Java

```
 // Java program to find minimum time
// required to produce m items.
import java.io.*;

public class GFG{

// Return the minimum time required to
// produce m items with given machines.
static int minTime(int []arr, int n,
```

```
                  int m)
{

    // Intialise time, items equal to 0.
    int t = 0;

    while (true)
    {
        int items = 0;

        // Calculating items at each second
        for (int i = 0; i < n; i++)
            items += (t / arr[i]);

        // If items equal to m return time.
        if (items >= m)
            return t;

        t++; // Increment time
    }
}

    // Driver Code
    static public void main (String[] args)
    {
            int []arr = { 1, 2, 3 };
            int n = arr.length;
            int m = 11;

    System.out.println(minTime(arr, n, m));
    }
}

// This code is contributed by vt_m.
```

C#

```
 // C# program to find minimum time
// required to produce m items.
using System;

public class GFG{

// Return the minimum time
// required to produce m
// items with given machines.
static int minTime(int []arr, int n,
                  int m)
```

```
{

    // Intialise time, items equal to 0.
    int t = 0;

    while (true)
    {
        int items = 0;

        // Calculating items at each second
        for (int i = 0; i < n; i++)
            items += (t / arr[i]);

        // If items equal to m return time.
        if (items >= m)
            return t;

        t++; // Increment time
    }
}

// Driven Code
static public void Main (String []args)
{
    int []arr = {1, 2, 3};
    int n = arr.Length;
    int m = 11;

    // Calling function
    Console.WriteLine(minTime(arr, n, m));

}
}
```

PHP

```
 <?php
// PHP program to find minimum time
// required to produce m items.

// Return the minimum time required to
// produce m items with given machines.
function minTime($arr, $n, $m)
{

    // Intialise time, items
    // equal to 0.
    $t = 0;
```

```
    while (1)
    {
        $items = 0;

        // Calculating items at each second
        for ( $i = 0; $i < $n; $i++)
            $items += ($t / $arr[$i]);

        // If items equal to m return time.
        if ($items >= $m)
            return $t;

        $t++; // Increment time
    }
}

    // Driver Code
    $arr = array(1, 2, 3);
    $n =count($arr);
    $m = 11;
    echo minTime($arr, $n, $m);

// This code is contributed by anuj_67.
?>
```

Output:

```
6
```

Method 2 (efficient):
The idea is to use Binary Search. Maximum possible time required to produce m items will be maximum time in the array, a_{max}, multiplied by m i.e $\mathbf{a_{max} * m}$. So, use binary search between 1 to a_{max} * m and find the minimum time which produce m items.
Below is the implementation of the above idea :

C++

```
 // Efficient C++ program to find minimum time
// required to produce m items.
#include<bits/stdc++.h>
using namespace std;

// Return the number of items can be
// produced in temp sec.
int findItems(int arr[], int n, int temp)
{
```

```
    int ans = 0;
    for (int i = 0; i < n; i++)
        ans += (temp/arr[i]);
    return ans;
}

// Binary search to find minimum time required
// to produce M items.
int bs(int arr[], int n, int m, int high)
{
    int low = 1;

    // Doing binary search to find minimum
    // time.
    while (low < high)
    {
        // Finding the middle value.
        int mid = (low+high)>>1;

        // Calculate number of items to
        // be produce in mid sec.
        int itm = findItems(arr, n, mid);

        // If items produce is less than
        // required, set low = mid + 1.
        if (itm < m)
            low = mid+1;

        //  Else set high = mid.
        else
            high = mid;
    }

    return high;
}

// Return the minimum time required to
// produce m items with given machine.
int minTime(int arr[], int n, int m)
{
    int maxval = INT_MIN;

    // Finding the maximum time in the array.
    for (int i = 0; i < n; i++)
        maxval = max(maxval, arr[i]);

    return bs(arr, n, m, maxval*m);
}
```

```
// Driven Program
int main()
{
    int arr[] = { 1, 2, 3 };
    int n = sizeof(arr)/sizeof(arr[0]);
    int m = 11;

    cout << minTime(arr, n, m) << endl;

    return 0;
}
```

Java

```
 // Efficient Java program to find
// minimum time required to
// produce m items.
import java.io.*;

public class GFG{

// Return the number of items can
// be produced in temp sec.
static int findItems(int []arr, int n,
                     int temp)
{
    int ans = 0;
    for (int i = 0; i < n; i++)
        ans += (temp / arr[i]);
    return ans;
}

// Binary search to find minimum time
// required to produce M items.
static int bs(int []arr, int n,
              int m, int high)

{
    int low = 1;

    // Doing binary search to
    // find minimum time.
    while (low < high)
    {
        // Finding the middle value.
        int mid = (low + high) >> 1;
```

```
        // Calculate number of items to
        // be produce in mid sec.
        int itm = findItems(arr, n, mid);

        // If items produce is less than
        // required, set low = mid + 1.
        if (itm < m)
            low = mid + 1;

        // Else set high = mid.
        else
            high = mid;
    }

    return high;
}

// Return the minimum time required to
// produce m items with given machine.
static int minTime(int []arr, int n,
                   int m)
{
    int maxval = Integer.MAX_VALUE;

    // Finding the maximum time in the array.
    for (int i = 0; i < n; i++)
        maxval = Math.max(maxval, arr[i]);

    return bs(arr, n, m, maxval * m);
}

// Driven Program
static public void main (String[] args)
{
    int []arr = {1, 2, 3};
        int n = arr.length;
    int m = 11;

    System.out.println(minTime(arr, n, m));
}
}

// This code is contributed by vt_m.
```

C#

```
 // Efficient C# program to find
// minimum time required to
```

```
// produce m items.
using System;

public class GFG{

// Return the number of items can
// be produced in temp sec.
static int findItems(int []arr, int n,
                     int temp)
{
    int ans = 0;
    for (int i = 0; i < n; i++)
        ans += (temp / arr[i]);
    return ans;
}

// Binary search to find minimum time
// required to produce M items..
static int bs(int []arr, int n,
              int m, int high)
{
    int low = 1;

    // Doing binary search to
    // find minimum time.
    while (low < high)
    {
        // Finding the middle value.
        int mid = (low + high) >> 1;

        // Calculate number of items to
        // be produce in mid sec.
        int itm = findItems(arr, n, mid);

        // If items produce is less than
        // required, set low = mid + 1.
        if (itm < m)
            low = mid + 1;

        // Else set high = mid.
        else
            high = mid;
    }

    return high;
}

// Return the minimum time required to
```

```
// produce m items with given machine.
static int minTime(int []arr, int n,
                   int m)
{
    int maxval = int.MinValue;

    // Finding the maximum time in the array.
    for (int i = 0; i < n; i++)
        maxval = Math.Max(maxval, arr[i]);

    return bs(arr, n, m, maxval * m);
}

// Driver Code
static public void Main ()
{
    int []arr = {1, 2, 3};
    int n = arr.Length;
    int m = 11;

    Console.WriteLine(minTime(arr, n, m));
}
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// Efficient PHP program to find minimum
// time required to produce m items.

// Return the number of items can be
// produced in temp sec.
function findItems( $arr, $n, $temp)
{
    $ans = 0;
    for($i = 0; $i < $n; $i++)
        $ans += ($temp / $arr[$i]);
    return $ans;
}

// Binary search to find minimum
// time required to produce M items.
function bs($arr, $n, $m, $high)
{
    $low = 1;
```

```
    // Doing binary search to
    // find minimum time.
    while ($low < $high)
    {

        // Finding the middle value.
        $mid = ($low + $high) >> 1;

        // Calculate number of items to
        // be produce in mid sec.
        $itm = findItems($arr, $n, $mid);

        // If items produce is less than
        // required, set low = mid + 1.
        if ($itm < $m)
            $low = $mid + 1;

        // Else set high = mid.
        else
            $high = $mid;
    }

    return $high;
}

// Return the minimum time required to
// produce m items with given machine.
function minTime($arr, $n, $m)
{
    $maxval = PHP_INT_MIN;

    // Finding the maximum
    // time in the array.
    for($i = 0; $i < $n; $i++)
        $maxval = max($maxval, $arr[$i]);

    return bs($arr, $n, $m, $maxval*$m);
}

    // Driver Code
    $arr = array(1, 2, 3);
    $n = count($arr);
    $m = 11;

    echo minTime($arr, $n, $m);

// This code is contributed by anuj_67.
?>
```

Output:

```
6
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/minimum-time-required-produce-m-items/

Chapter 179

Minimum value of “max + min” in a subarray

Minimum value of ”max + min” in a subarray - GeeksforGeeks

Given a array of n positive elements we need to find the lowest possible sum of max and min elements in a subarray given that size of subarray should be greater than equal to 2.

Examples:

```
Input : arr[] = {1 12 2 2}
Output : 4
Sum of 2+2 of subarray [2, 2]

Input  : arr[] = {10 20 30 40 23 45}
Output : 30
Sum of 10+20 of subarray[10, 20]
```

A **simple solution** is to generate all subarrays, compute sum of maximum and minimum and finally return lowest sum.

An **efficient solution** is based on the fact that adding any element to a subarray would not increase sum of maximum and minimum. Consider the array [a1, a2, a3, a4, a5….an] Each ai will be minimum of some subarray [al, ar] such that i lies between [l, r] and all elements in the subarray are greater than or equal to ai. The cost of such subarray would be ai + max(subarray). Since the max of an array will never decrease on adding elements to the array, It will only increase if we add larger elements so It is always optimal to consider only those subarrays having length 2.

In short consider all subarrays of length 2 and compare sum and take the minimum one which will reduce the time complexity by O(N) now we have to run the loop only once.

C++

```
 // CPP program to find sum of maximum and
// minimum in any subarray of an array of
// positive numbers.
#include <bits/stdc++.h>
using namespace std;

int maxSum(int arr[], int n)
{
    if (n < 2)
        return -1;
    int ans = arr[0] + arr[1];
    for (int i = 1; i + 1 < n; i++)
        ans = min(ans, (arr[i] + arr[i + 1]));
    return ans;
}

// Driver code
int main()
{
    int arr[] = {1, 12, 2, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << maxSum(arr, n);
    return 0;
}
```

Java

```
 // java program to find sum of maximum and
// minimum in any subarray of an array of
// positive numbers.
import java.io.*;

class GFG {

    static int maxSum(int arr[], int n)
    {
        if (n < 2)
            return -1;
        int ans = arr[0] + arr[1];
        for (int i = 1; i + 1 < n; i++)
            ans = Math.min(ans, (arr[i]
                            + arr[i + 1]));
        return ans;
    }

    // Driver code
    public static void main (String[] args)
    {
```

```
        int arr[] = {1, 12, 2, 2};
        int n = arr.length;

        System.out.println( maxSum(arr, n));
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // C# program to find sum of maximum and
// minimum in any subarray of an array of
// positive numbers.
using System ;

class GFG {

    static int maxSum(int []arr, int n)
    {
        if (n < 2)
            return -1;
        int ans = arr[0] + arr[1];
        for (int i = 1; i + 1 < n; i++)
            ans = Math.Min(ans, (arr[i]
                        + arr[i + 1]));
        return ans;
    }

    // Driver code
    public static void Main ()
    {
        int []arr = {1, 12, 2, 2};
        int n = arr.Length;

        Console.WriteLine( maxSum(arr, n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find sum of
// maximum and minimum in any
// subarray of an array of
```

```
// positive numbers.

function maxSum( $arr, $n)
{
    if ($n < 2)
        return -1;
    $ans = $arr[0] + $arr[1];
    for ( $i = 1; $i + 1 < $n; $i++)
        $ans = min($ans, ($arr[$i] +
                    $arr[$i + 1]));
    return $ans;
}

    // Driver code
    $arr = array(1, 12, 2, 2);
    $n = count($arr);
    echo maxSum($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
4
```

Time Complexity : O(n)
Auxiliary Space : O(1)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/minimum-value-max-min-subarray/

Chapter 180

Most frequent element in an array

Most frequent element in an array - GeeksforGeeks

Given an array, find the most frequent element in it. If there are multiple elements that appear maximum number of times, print any one of them.

Examples:

```
Input : arr[] = {1, 3, 2, 1, 4, 1}
Output : 1
1 appears three times in array which
is maximum frequency.

Input : arr[] = {10, 20, 10, 20, 30, 20, 20}
Output : 20
```

A **simple solution** is to run two loops. The outer loop picks all elements one by one. The inner loop finds frequency of the picked element and compares with the maximum so far. Time complexity of this solution is $O(n^2)$

A **better solution** is to do sorting. We first sort the array, then linearly traverse the array.

C++

```
 // CPP program to find the most frequent element
// in an array.
#include <bits/stdc++.h>
using namespace std;

int mostFrequent(int arr[], int n)
{
```

```
    // Sort the array
    sort(arr, arr + n);

    // find the max frequency using linear traversal
    int max_count = 1, res = arr[0], curr_count = 1;
    for (int i = 1; i < n; i++) {
        if (arr[i] == arr[i - 1])
            curr_count++;
        else {
            if (curr_count > max_count) {
                max_count = curr_count;
                res = arr[i - 1];
            }
            curr_count = 1;
        }
    }

    // If last element is most frequent
    if (curr_count > max_count)
    {
        max_count = curr_count;
        res = arr[n - 1];
    }

    return res;
}

// driver program
int main()
{
    int arr[] = { 1, 5, 2, 1, 3, 2, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << mostFrequent(arr, n);
    return 0;
}
```

Java

```
 //Java program to find the most frequent element
//in an array
import java.util.*;

class GFG {

    static int mostFrequent(int arr[], int n)
    {

        // Sort the array
```

```
        Arrays.sort(arr);

        // find the max frequency using linear
        // traversal
        int max_count = 1, res = arr[0];
        int curr_count = 1;

        for (int i = 1; i < n; i++)
        {
            if (arr[i] == arr[i - 1])
                curr_count++;
            else
            {
                if (curr_count > max_count)
                {
                    max_count = curr_count;
                    res = arr[i - 1];
                }
                curr_count = 1;
            }
        }

        // If last element is most frequent
        if (curr_count > max_count)
        {
            max_count = curr_count;
            res = arr[n - 1];
        }

        return res;
    }

    // Driver program
    public static void main (String[] args) {

        int arr[] = {1, 5, 2, 1, 3, 2, 1};
        int n = arr.length;

        System.out.println(mostFrequent(arr,n));

    }
}

// This code is contributed by Akash Singh.
```

Python3

```
 # Python3 program to find the most
```

```
# frequent element in an array.

def mostFrequent(arr, n):

    # Sort the array
    arr.sort()

    # find the max frequency using
    # linear traversal
    max_count = 1; res = arr[0]; curr_count = 1

    for i in range(1, n):
        if (arr[i] == arr[i - 1]):
            curr_count += 1

        else :
            if (curr_count > max_count):
                max_count = curr_count
                res = arr[i - 1]

            curr_count = 1

    # If last element is most frequent
    if (curr_count > max_count):

        max_count = curr_count
        res = arr[n - 1]

    return res

# Driver Code
arr = [1, 5, 2, 1, 3, 2, 1]
n = len(arr)
print(mostFrequent(arr, n))

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# program to find the most
// frequent element in an array
using System;

class GFG {

    static int mostFrequent(int []arr, int n)
    {
```

```
        // Sort the array
        Array.Sort(arr);

        // find the max frequency using
        // linear traversal
        int max_count = 1, res = arr[0];
        int curr_count = 1;

        for (int i = 1; i < n; i++)
        {
            if (arr[i] == arr[i - 1])
                curr_count++;
            else
            {
                if (curr_count > max_count)
                {
                    max_count = curr_count;
                    res = arr[i - 1];
                }
                curr_count = 1;
            }
        }

        // If last element is most frequent
        if (curr_count > max_count)
        {
            max_count = curr_count;
            res = arr[n - 1];
        }

        return res;
    }

    // Driver code
    public static void Main ()
    {

        int []arr = {1, 5, 2, 1, 3, 2, 1};
        int n = arr.Length;

        Console.WriteLine(mostFrequent(arr,n));

    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find the
// most frequent element
// in an array.

function mostFrequent( $arr, $n)
{

    // Sort the array
    sort($arr);
    sort($arr , $n);

    // find the max frequency
    // using linear traversal
    $max_count = 1;
    $res = $arr[0];
    $curr_count = 1;
    for ($i = 1; $i < $n; $i++)
    {
        if ($arr[$i] == $arr[$i - 1])
            $curr_count++;
        else
        {
            if ($curr_count > $max_count)
            {
                $max_count = $curr_count;
                $res = $arr[$i - 1];
            }
            $curr_count = 1;
        }
    }

    // If last element
    // is most frequent
    if ($curr_count > $max_count)
    {
        $max_count = $curr_count;
        $res = $arr[$n - 1];
    }

    return $res;
}

// Driver Code
{
    $arr = array(1, 5, 2, 1, 3, 2, 1);
    $n = sizeof($arr) / sizeof($arr[0]);
    echo mostFrequent($arr, $n);
```

```
    return 0;
}

// This code is contributed by nitin mittal
?>
```

Output :

```
1
```

Time Complexity : O(n Log n)
Auxiliary Space : O(1)

An **efficient solution** is to use hashing. We create a hash table and store elements and their frequency counts as key value pairs. Finally we traverse the hash table and print the key with maximum value.

C++

```
 // CPP program to find the most frequent element
// in an array.
#include <bits/stdc++.h>
using namespace std;

int mostFrequent(int arr[], int n)
{
    // Insert all elements in hash.
    unordered_map<int, int> hash;
    for (int i = 0; i < n; i++)
        hash[arr[i]]++;

    // find the max frequency
    int max_count = 0, res = -1;
    for (auto i : hash) {
        if (max_count < i.second) {
            res = i.first;
            max_count = i.second;
        }
    }

    return res;
}

// driver program
int main()
{
```

```
    int arr[] = { 1, 5, 2, 1, 3, 2, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << mostFrequent(arr, n);
    return 0;
}
```

Java

```
 //Java program to find the most frequent element
//in an array
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

class GFG {

    static int mostFrequent(int arr[], int n)
    {

        // Insert all elements in hash
        Map<Integer, Integer> hp =
               new HashMap<Integer, Integer>();

        for(int i = 0; i < n; i++)
        {
            int key = arr[i];
            if(hp.containsKey(key))
            {
                int freq = hp.get(key);
                freq++;
                hp.put(key, freq);
            }
            else
            {
                hp.put(key, 1);
            }
        }

        // find max frequency.
        int max_count = 0, res = -1;

        for(Entry<Integer, Integer> val : hp.entrySet())
        {
            if (max_count < val.getValue())
            {
                res = val.getKey();
                max_count = val.getValue();
            }
```

```
        }

        return res;
    }

    // Driver code
    public static void main (String[] args) {

        int arr[] = {1, 5, 2, 1, 3, 2, 1};
        int n = arr.length;

        System.out.println(mostFrequent(arr, n));
    }
}

// This code is contributed by Akash Singh.
```

C#

```
 // C# program to find the most
// frequent element in an array
using System;
using System.Collections.Generic;

class GFG
{
    static int mostFrequent(int []arr,
                            int n)
    {
        // Insert all elements in hash
        Dictionary<int, int> hp =
                    new Dictionary<int, int>();

        for (int i = 0; i < n; i++)
        {
            int key = arr[i];
            if(hp.ContainsKey(key))
            {
                int freq = hp[key];
                freq++;
                hp[key] = freq;
            }
            else
                hp.Add(key, 1);
        }

        // find max frequency.
        int min_count = 0, res = -1;
```

```
        foreach (KeyValuePair<int,
                    int> pair in hp)
        {
            if (min_count < pair.Value)
            {
                res = pair.Key;
                min_count = pair.Value;
            }
        }
        return res;
    }

    // Driver code
    static void Main ()
    {
        int []arr = new int[]{1, 5, 2,
                              1, 3, 2, 1};
        int n = arr.Length;

        Console.Write(mostFrequent(arr, n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

```
1
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Improved By : vt_m, nitin mittal, dipesh_jain, manishshaw1

Source

https://www.geeksforgeeks.org/frequent-element-array/

Chapter 181

Move all occurrences of an element to end in a linked list

Move all occurrences of an element to end in a linked list - GeeksforGeeks

Given a linked list and a key in it, the task is to move all occurrences of given key to end of linked list, keeping order of all other elements same.

Examples:

```
Input  : 1 -> 2 -> 2 -> 4 -> 3
         key = 2
Output : 1 -> 4 -> 3 -> 2 -> 2

Input  : 6 -> 6 -> 7 -> 6 -> 3 -> 10
         key = 6
Output : 7 -> 3 -> 10 -> 6 -> 6 -> 6
```

A **simple solution** is to one by one find all occurrences of given key in linked list. For every found occurrence, insert it at the end. We do it till all occurrences of given key are moved to end.

Time Complexity : $O(n^2)$

Efficient Solution 1 : is to keep two pointers:
pCrawl => Pointer to traverse the whole list one by one.
pKey => Pointer to an occurrence of key if a key is found. Else same as pCrawl.

We start both of the above pointers from head of linked list. We move **pKey** only when **pKey** is not pointing to a key. We always move **pCrawl**. So when **pCrawl** and **pKey** are not same, we must have found a key which lies before **pCrawl**, so we swap data of **pCrawl** and **pKey**, and move **pKey** to next location. The loop invariant is, after swapping of data, all elements from **pKey** to **pCrawl** are keys.

Below is the C++ implementation of this approach.

```
 // C++ program to move all occurrences of a
// given key to end.
#include<bits/stdc++.h>

// A Linked list Node
struct Node
{
    int data;
    struct Node* next;
};

// A urility function to create a new node.
struct Node *newNode(int x)
{
   Node *temp = new Node;
   temp->data = x;
   temp->next = NULL;
}

// Utility function to print the elements
// in Linked list
void printList(Node *head)
{
    struct Node* temp = head;
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Moves all occurrences of given key to
// end of linked list.
void moveToEnd(Node *head, int key)
{
    // Keeps track of locations where key
    // is present.
    struct Node *pKey = head;

    // Traverse list
    struct Node *pCrawl = head;
    while (pCrawl != NULL)
    {
        // If current pointer is not same as pointer
        // to a key location, then we must have found
        // a key in linked list. We swap data of pCrawl
        // and pKey and move pKey to next position.
```

```
        if (pCrawl != pKey && pCrawl->data != key)
        {
            pKey->data = pCrawl->data;
            pCrawl->data = key;
            pKey = pKey->next;
        }

        // Find next position where key is present
        if (pKey->data != key)
            pKey = pKey->next;

        // Moving to next Node
        pCrawl = pCrawl->next;
    }
}

// Driver code
int main()
{
    Node *head = newNode(10);
    head->next = newNode(20);
    head->next->next = newNode(10);
    head->next->next->next = newNode(30);
    head->next->next->next->next = newNode(40);
    head->next->next->next->next->next = newNode(10);
    head->next->next->next->next->next->next = newNode(60);

    printf("Before moveToEnd(), the Linked list is\n");
    printList(head);

    int key = 10;
    moveToEnd(head, key);

    printf("\nAfter moveToEnd(), the Linked list is\n");
    printList(head);

    return 0;
}
```

Output:

```
Before moveToEnd(), the Linked list is
10 20 10 30 40 10 60

After moveToEnd(), the Linked list is
20 30 40 60 10 10 10
```

Time Complexity : O(n) requires only one traversal of list.

Efficient Solution 2 :
1. Traverse the linked list and take a pointer at tail.
2. Now, check for the key and node->data, if they are equal, move the node to last-next, else move
ahead.

Java

```
 // Java code to remove key element to end of linked list
import java.util.*;

// Node class
class Node{
    int data;
    Node next;

    public Node(int data){
        this.data=data;
        this.next=null;
    }
}

class gfg{

    static Node root;

    // Function to remove key to end
    public static Node keyToEnd(Node head, int key){

        // Node to keep pointing to tail
        Node tail = head;

        if(head==null){
            return null;
        }

        while(tail.next !=null){
            tail=tail.next;
        }

        // Node to point to last of linked list
        Node last = tail;

        Node current = head;
        Node prev = null;

        // Node prev2 to point to previous when head.data!=key
        Node prev2 = null;
```

```
    // loop to perform operations to remove key to end
    while(current!=tail){
        if(current.data==key && prev2 == null){
            prev = current;
            current = current.next;
            head = current;
            last.next = prev;
            last = last.next;
            last.next = null;
            prev = null;
        }
        else{
            if(current.data==key && prev2!=null){
                prev = current;
                current = current.next;
                prev2.next = current;
                last.next = prev;
                last = last.next;
                last.next = null;
            }
            else
            if(current != tail){
                prev2 = current;
                current = current.next;
            }
        }
    }
    return head;
}

// Function to display linked list
public static void display(Node root){
    while(root!=null){
        System.out.print(root.data+" ");
        root = root.next;
    }
}

// Driver Code
public static void main(String args[]){
    root = new Node(5);
    root.next = new Node(2);
    root.next.next = new Node(2);
    root.next.next.next = new Node(7);
    root.next.next.next.next = new Node(2);
    root.next.next.next.next.next = new Node(2);
    root.next.next.next.next.next.next = new Node(2);
```

```
        int key = 2;
        System.out.println("Linked List before operations :");
        display(root);
        System.out.println("\nLinked List after operations :");
        root = keyToEnd(root, key);
        display(root);
    }
}
```

Thanks to **Ravinder Kumar** for suggesting this method.

Efficient Solution 3 : is to maintain a separate list of keys. We initialize this list of keys as empty. We traverse given list. For every key found, we remove it from the original list and insert into the separate list of keys. We finally link list of keys at the end of remaining given list. Time complexity of this solution is also O(n) and it also requires only one traversal of list.

Source

https://www.geeksforgeeks.org/move-occurrences-element-end-linked-list/

Chapter 182

N/3 repeated number in an array with O(1) space

N/3 repeated number in an array with O(1) space - GeeksforGeeks

We are given a read only array of n integers. Find any element that appears more than n/3 times in the array in linear time and constant additional space. If no such element exists, return -1.

Examples:

```
Input : [10, 10, 20, 30, 10, 10]
Output : 10
10 occurs 4 times which is more than 6/3.

Input : [20, 30, 10, 10, 5, 4, 20, 1, 2]
Output : -1
```

The idea is based on Moore's Voting algorithm.
We first find two candidates. Then we check if any of these two candidates is actually a majority. Below is the solution for above approach.

C++

```
 // CPP program to find if any element appears
// more than n/3.
#include <bits/stdc++.h>
using namespace std;

int appearsNBy3(int arr[], int n)
{
```

```
int count1 = 0, count2 = 0;
int first=INT_MAX    , second=INT_MAX    ;

for (int i = 0; i < n; i++) {

    // if this element is previously seen,
    // increment count1.
    if (first == arr[i])
        count1++;

    // if this element is previously seen,
    // increment count2.
    else if (second == arr[i])
        count2++;

    else if (count1 == 0) {
        count1++;
        first = arr[i];
    }

    else if (count2 == 0) {
        count2++;
        second = arr[i];
    }

    // if current element is different from
    // both the previously seen variables,
    // decrement both the counts.
    else {
        count1--;
        count2--;
    }
}

count1 = 0;
count2 = 0;

// Again traverse the array and find the
// actual counts.
for (int i = 0; i < n; i++) {
    if (arr[i] == first)
        count1++;

    else if (arr[i] == second)
        count2++;
}

if (count1 > n / 3)
```

```
        return first;

    if (count2 > n / 3)
        return second;

    return -1;
}

int main()
{
    int arr[] = { 1, 2, 3, 1, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << appearsNBy3(arr, n) << endl;
    return 0;
}
```

Java

```
 // Java program to find if any element appears
// more than n/3.
class GFG {

    static int appearsNBy3(int arr[], int n)
    {
        int count1 = 0, count2 = 0;

        // take the integers as the maximum
        // value of integer hoping the integer
        // would not be present in the array
        int first = Integer.MAX_VALUE;
        int second = Integer.MAX_VALUE;

        for (int i = 1; i < n; i++) {

            // if this element is previously
            // seen, increment count1.
            if (first == arr[i])
                count1++;

            // if this element is previously
            // seen, increment count2.
            else if (second == arr[i])
                count2++;

            else if (count1 == 0) {
                count1++;
                first = arr[i];
            }
```

```
            else if (count2 == 0) {
                count2++;
                second = arr[i];
            }

            // if current element is different
            // from both the previously seen
            // variables, decrement both the
            // counts.
            else {
                count1--;
                count2--;
            }
        }

        count1 = 0;
        count2 = 0;

        // Again traverse the array and
        // find the actual counts.
        for (int i = 0; i < n; i++) {
            if (arr[i] == first)
                count1++;

            else if (arr[i] == second)
                count2++;
        }

        if (count1 > n / 3)
            return first;

        if (count2 > n / 3)
            return second;

        return -1;
    }

    // Driver code
    public static void main(String args[])
    {
        int arr[] = { 1, 2, 3, 1, 1 };
        int n = arr.length;
        System.out.println(appearsNBy3(arr, n));

    }
}
```

```
// This code is contributed by Arnab Kundu
```

Python 3

```
 # Python 3 program to find if
# any element appears more than
# n/3.
import sys

def appearsNBy3(arr, n):

    count1 = 0
    count2 = 0
    first = sys.maxsize
    second = sys.maxsize

    for i in range(0, n):

        # if this element is
        # previously seen,
        # increment count1.
        if (first == arr[i]):
            count1 += 1

        # if this element is
        # previously seen,
        # increment count2.
        elif (second == arr[i]):
            count2 += 1

        elif (count1 == 0):
            count1 += 1
            first = arr[i]

        elif (count2 == 0):
            count2 += 1
            second = arr[i]


        # if current element is
        # different from both
        # the previously seen
        # variables, decrement
        # both the counts.
        else:
            count1 -= 1
            count2 -= 1
```

```
    count1 = 0
    count2 = 0

    # Again traverse the array
    # and find the actual counts.
    for i in range(0, n):
        if (arr[i] == first):
            count1 += 1

        elif (arr[i] == second):
            count2 += 1


    if (count1 > n / 3):
        return first

    if (count2 > n / 3):
        return second

    return -1

# Driver code
arr = [1, 2, 3, 1, 1 ]
n = len(arr)
print(appearsNBy3(arr, n))

# This code is contributed by
# Smitha
```

C#

```
 // C# program to find if any element appears
// more than n/3.
using System;

class GFG {

    static int appearsNBy3(int []arr, int n)
    {
        int count1 = 0, count2 = 0;

        // take the integers as the maximum
        // value of integer hoping the integer
        // would not be present in the array
        int first = int.MaxValue;
        int second = int.MaxValue;
```

```
for (int i = 1; i < n; i++) {

    // if this element is previously
    // seen, increment count1.
    if (first == arr[i])
        count1++;

    // if this element is previously
    // seen, increment count2.
    else if (second == arr[i])
        count2++;

    else if (count1 == 0) {
        count1++;
        first = arr[i];
    }

    else if (count2 == 0) {
        count2++;
        second = arr[i];
    }

    // if current element is different
    // from both the previously seen
    // variables, decrement both the
    // counts.
    else {
        count1--;
        count2--;
    }
}

count1 = 0;
count2 = 0;

// Again traverse the array and
// find the actual counts.
for (int i = 0; i < n; i++) {
    if (arr[i] == first)
        count1++;

    else if (arr[i] == second)
        count2++;
}

if (count1 > n / 3)
    return first;
```

```
        if (count2 > n / 3)
            return second;

        return -1;
    }

    // Driver code
    static public void Main(String []args)
    {
        int []arr = { 1, 2, 3, 1, 1 };
        int n = arr. Length;
        Console.WriteLine(appearsNBy3(arr, n));
    }
}

// This code is contributed by Arnab Kundu
```

PHP

```
 <?php
// PHP program to find if any element appears
// more than n/3.

function appearsNBy3( $arr,  $n)
{
     $count1 = 0; $count2 = 0;
    $first = PHP_INT_MAX ; $second = PHP_INT_MAX ;

    for ( $i = 0; $i < $n; $i++) {

        // if this element is previously seen,
        // increment count1.
        if ($first == $arr[$i])
            $count1++;

        // if this element is previously seen,
        // increment count2.
        else if ($second == $arr[$i])
            $count2++;

        else if ($count1 == 0) {
            $count1++;
            $first = $arr[$i];
        }

        else if ($count2 == 0) {
            $count2++;
```

```
            $second = $arr[$i];
        }

        // if current element is different from
        // both the previously seen variables,
        // decrement both the counts.
        else {
            $count1--;
            $count2--;
        }
    }

    $count1 = 0;
    $count2 = 0;

    // Again traverse the array and find the
    // actual counts.
    for ($i = 0; $i < $n; $i++) {
        if ($arr[$i] == $first)
            $count1++;

        else if ($arr[$i] == $second)
            $count2++;
    }

    if ($count1 > $n / 3)
        return $first;

    if ($count2 > $n / 3)
        return $second;

    return -1;
}

// Driver code
$arr = array( 1, 2, 3, 1, 1 );
$n = count($arr);
echo appearsNBy3($arr, $n) ;

// This code is contributed by anuj_67.
?>
```

Output:

```
1
```

Improved By : andrew1234, Smitha Dinesh Semwal, vt_m

Source

https://www.geeksforgeeks.org/n3-repeated-number-array-o1-space/

Chapter 183

Next Smaller Element

Next Smaller Element - GeeksforGeeks

Given an array, print the Next Smaller Element (NSE) for every element. The Smaller smaller Element for an element x is the first smaller element on the right side of x in array. Elements for which no smaller element exist (on right side), consider next smaller element as -1.

Examples:
a) For any array, rightmost element always has next smaller element as -1.
b) For an array which is sorted in increasing order, all elements have next smaller element as -1.
c) For the input array **[4, 8, 5, 2, 25}**, the next smaller elements for each element are as follows.

```
Element       NSE
   4      -->   2
   8      -->   5
   5      -->   2
   2      -->   -1
   25     -->   -1
```

d) For the input array **[13, 7, 6, 12},** the next smaller elements for each element are as follows.

```
  Element        NSE
   13      -->    7
   7       -->    6
   6       -->    -1
   12     -->     -1
```

Method 1 (Simple)
Use two loops: The outer loop picks all the elements one by one. The inner loop looks for the first smaller element for the element picked by outer loop. If a smaller element is found then that element is printed as next, otherwise -1 is printed.

Thanks to Sachin for providing following code.

C

```
// Simple C program to print next smaller elements
// in a given array
#include<stdio.h>

/* prints element and NSE pair for all elements of
arr[] of size n */
void printNSE(int arr[], int n)
{
    int next, i, j;
    for (i=0; i<n; i++)
    {
        next = -1;
        for (j = i+1; j<n; j++)
        {
            if (arr[i] > arr[j])
            {
                next = arr[j];
                break;
            }
        }
        printf("%d -- %d\n", arr[i], next);
    }
}

int main()
{
    int arr[]= {11, 13, 21, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    printNSE(arr, n);
    return 0;
}
```

Java

```
 // Simple Java program to print next
// smaller elements in a given array

class Main {
```

```
    /* prints element and NSE pair for
     all elements of arr[] of size n */
    static void printNSE(int arr[], int n)
    {
        int next, i, j;
        for (i = 0; i < n; i++) {
            next = -1;
            for (j = i + 1; j < n; j++) {
                if (arr[i] > arr[j]) {
                    next = arr[j];
                    break;
                }
            }
            System.out.println(arr[i] + " -- " + next);
        }
    }

    public static void main(String args[])
    {
        int arr[] = { 11, 13, 21, 3 };
        int n = arr.length;
        printNSE(arr, n);
    }
}
```

Python

```
# Function to print element and NSE pair for all elements of list
def printNSE(arr):

    for i in range(0, len(arr), 1):

        next = -1
        for j in range(i + 1, len(arr), 1):
            if arr[i] > arr[j]:
                next = arr[j]
                break

        print(str(arr[i]) + " -- " + str(next))

# Driver program to test above function
arr = [11, 13, 21, 3]
printNSE(arr)

# This code is contributed by Sunny Karira
```

C#

```
 // Simple C# program to print next
// smaller elements in a given array
using System;

class GFG {

    /* prints element and NSE pair for
    all elements of arr[] of size n */
    static void printNSE(int[] arr, int n)
    {
        int next, i, j;
        for (i = 0; i < n; i++) {
            next = -1;
            for (j = i + 1; j < n; j++) {
                if (arr[i] > arr[j]) {
                    next = arr[j];
                    break;
                }
            }
            Console.WriteLine(arr[i] + " -- " + next);
        }
    }

    // driver code
    public static void Main()
    {
        int[] arr = { 11, 13, 21, 3 };
        int n = arr.Length;

        printNSE(arr, n);
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// Simple PHP program to print next
// smaller elements in a given array

/* prints element and NSE pair for
   all elements of arr[] of size n */
function printNSE($arr, $n)
{
    for ($i = 0; $i < $n; $i++)
    {
        $next = -1;
```

```
        for ($j = $i + 1; $j < $n; $j++)
        {
            if ($arr[$i] > $arr[$j])
            {
                $next = $arr[$j];
                break;
            }
        }
        echo $arr[$i]." -- ". $next."\n";

    }
}

    // Driver Code
    $arr= array(11, 13, 21, 3);
    $n = count($arr);
    printNSE($arr, $n);

// This code is contributed by Sam007
?>
```

Output:

```
11 -- 3
13 -- 3
21 -- 3
3 -- -1
```

Time Complexity: O(n^2). The worst case occurs when all elements are sorted in decreasing order.

Method 2 (Using Stack)
This problem is similar to next greater element. Here we maintain items in increasing order in the stack (instead of decreasing in next greater element problem).

1) Push the first element to stack.
2) Pick rest of the elements one by one and follow following steps in loop.
….a) Mark the current element as *next*.
….b) If stack is not empty, then pop an element from stack and compare it with *next*.
….c) If next is smaller than the popped element, then *next* is the next smaller element for the popped element.
….d) Keep popping from the stack while the popped element is greater than *next*. *next* becomes the next smaller element for all such popped elements
3) After the loop in step 2 is over, pop all the elements from stack and print -1 as next element for them.

C++

```
 // A Stack based C++ program to find next
// smaller element for all array elements.
#include <bits/stdc++.h>

using namespace std;

/* prints element and NSE pair for all
elements of arr[] of size n */
void printNSE(int arr[], int n)
{
    stack<int> s;

    /* push the first element to stack */
    s.push(arr[0]);

    // iterate for rest of the elements
    for (int i = 1; i < n; i++) {

        if (s.empty()) {
            s.push(arr[i]);
            continue;
        }

        /* if stack is not empty, then
       pop an element from stack.
       If the popped element is smaller
       than next, then
    a) print the pair
    b) keep popping while elements are
    smaller and stack is not empty */
        while (s.empty() == false && s.top() > arr[i]) {
            cout << s.top() << " --> " << arr[i] << endl;
            s.pop();
        }

        /* push next to stack so that we can find
    next smaller for it */
        s.push(arr[i]);
    }

    /* After iterating over the loop, the remaining
  elements in stack do not have the next smaller
  element, so print -1 for them */
    while (s.empty() == false) {
        cout << s.top() << " --> " << -1 << endl;
        s.pop();
    }
}
```

```
/* Driver program to test above functions */
int main()
{
    int arr[] = { 11, 13, 21, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printNSE(arr, n);
    return 0;
}
```

Java

```
 // A Stack based Java program to find next
// smaller element for all array elements.
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
    /* prints element and NSE pair for all
    elements of arr[] of size n */
    public static void printNSE(int arr[], int n)
    {
        Stack<Integer> s = new Stack<Integer>();

        /* push the first element to stack */
        s.push(arr[0]);

        // iterate for rest of the elements
        for (int i = 1; i < n; i++) {

        if (s.empty()) {
            s.push(arr[i]);
            continue;
        }


        /* if stack is not empty, then
        pop an element from stack.
        If the popped element is smaller
        than next, then
        a) print the pair
        b) keep popping while elements are
        smaller and stack is not empty */
    while (s.empty() == false && s.peek() > arr[i]) {
            System.out.println(s.peek() + " --> " + arr[i]);
            s.pop();
```

```
        }

        /* push next to stack so that we can find
        next smaller for it */
        s.push(arr[i]);
        }

        /* After iterating over the loop, the remaining
        elements in stack do not have the next smaller
        element, so print -1 for them */
            while (s.empty() == false) {
        System.out.println(s.peek() + " --> " + "-1");
        s.pop();
        }
    }
    /* Driver program to test above functions */
    public static void main (String[] args) {
        int arr[] = { 11, 13, 21, 3};
        int n = arr.length;
        printNSE(arr, n);
    }
}
```

Output:

```
21 --> 3
13 --> 3
11 --> 3
3 --> -1
```

Time Complexity: O(n). The worst case occurs when all elements are sorted in increasing order. If elements are sorted in increasing order, then every element is processed at most 4 times.
a) Initially pushed to the stack.
b) Popped from the stack when next element is being processed.
c) Pushed back to the stack because next element is smaller.
d) Popped from the stack in step 3 of algo.

How to get elements in same order as input?
The above approach may not produce output elements in same order as input. To achieve same order, we can use an unordered_map in C++ (or HashMap in Java).

C++

```
 // A Stack based C++ program to find next
// smaller element for all array elements
```

```
// in same order as input.
#include <bits/stdc++.h>
using namespace std;

/* prints element and NSE pair for all
elements of arr[] of size n */
void printNSE(int arr[], int n)
{
    stack<int> s;
    unordered_map<int, int> mp;

    /* push the first element to stack */
    s.push(arr[0]);


    // iterate for rest of the elements
    for (int i = 1; i < n; i++) {

        if (s.empty()) {
            s.push(arr[i]);
            continue;
        }

        /* if stack is not empty, then
       pop an element from stack.
       If the popped element is smaller
       than next, then
    a) print the pair
    b) keep popping while elements are
    smaller and stack is not empty */
        while (s.empty() == false && s.top() > arr[i]) {
            mp[s.top()] = arr[i];
            s.pop();
        }

        /* push next to stack so that we can find
    next smaller for it */
        s.push(arr[i]);
    }

    /* After iterating over the loop, the remaining
  elements in stack do not have the next smaller
  element, so print -1 for them */
    while (s.empty() == false) {
         mp[s.top()] = -1;
        s.pop();
    }
```

```
    for (int i=0; i<n; i++)
       cout << arr[i] << " ---> " << mp[arr[i]] << endl;
}

/* Driver program to test above functions */
int main()
{
    int arr[] = { 11, 13, 21, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printNSE(arr, n);
    return 0;
}
```

Java

```
 // A Stack based Java program to find next
// smaller element for all array elements
// in same order as input.
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
    /* prints element and NSE pair for all
    elements of arr[] of size n */
public static void printNSE(int arr[], int n)
{
    Stack<Integer> s = new Stack<Integer>();
    HashMap<Integer,Integer> mp = new HashMap<Integer,Integer>();

    /* push the first element to stack */
    s.push(arr[0]);


    // iterate for rest of the elements
    for (int i = 1; i < n; i++) {

        if (s.empty()) {
            s.push(arr[i]);
            continue;
        }


        /* if stack is not empty, then
    pop an element from stack.
    If the popped element is smaller
    than next, then
```

```
    a) print the pair
    b) keep popping while elements are
    smaller and stack is not empty */

    while (s.empty() == false && s.peek() > arr[i]) {
            mp.put(s.peek(), arr[i]);
            s.pop();
        }

        /* push next to stack so that we can find
        next smaller for it */
        s.push(arr[i]);
    }



    /* After iterating over the loop, the remaining
    elements in stack do not have the next smaller
    element, so print -1 for them */
    while (s.empty() == false) {
        mp.put(s.peek(), -1);
        s.pop();
    }


    for (int i=0; i<n; i++)
    System.out.println(arr[i] + " ---> " + mp.get(arr[i]));
}

    /* Driver program to test above functions */
    public static void main (String[] args) {
        int arr[] = { 11, 13, 21, 3};
        int n = arr.length;
        printNSE(arr, n);
    }
}
```

Output:

```
11 ---> 3
13 ---> 3
21 ---> 3
3 ---> -1
```

Improved By : nabaneet247

Source

https://www.geeksforgeeks.org/next-smaller-element/

Chapter 184

No of pairs (a[j] >= a[i]) with k numbers in range (a[i], a[j]) that are divisible by x

No of pairs (a[j] >= a[i]) with k numbers in range (a[i], a[j]) that are divisible by x - GeeksforGeeks

Given an array and two numbers x and k. Find the number of different ordered pairs of indexes (i, j) such that a[j] >= a[i] and there are exactly k integers num such that num is divisible by x and num is in range a[i]-a[j].

Examples:

```
Input : arr[] = {1, 3, 5, 7}
        x = 2, k = 1
Output : 3
Explanation: The pairs (1, 3), (3, 5) and (5, 7)
have k (which is 1) integers i.e., 2, 4, 6
respectively for every pair in between them.

Input  : arr[] = {5, 3, 1, 7}
         x = 2, k = 0
Output : 4
Explanation: The pairs with indexes (1, 1), (2, 2),
(3, 3), (4, 4)  have k = 0 integers that are
divisible by 2 in between them.
```

A **naive approach** is to traverse through all pairs possible and count the number of pairs that have k integers in between them which are divisible by x.
Time complexity: O(n^2)

An **efficient approach** is to sort the array and use binary search to find out the right and left boundaries of numbers(use lower_bound function inbuilt function to do it) which satisfy the condition and which do not. We have to sort the array as it is given every pair should be a[j] >= a[i] irrespective of value of i and j. After sorting we traverse through n elements, and find the number with whose multiplication with x gives a[i]-1, so that we can find k number by adding k to d = a[i]-1/x. So we binary search for the value (d+k)*x to get the multiple with which we can make a pair of a[i] as it will have exactly k integers in between a[i] and a[j]. In this way we get the left boundary for a[j] using binary search in O(log n), and for all other pairs possible with a[i], we need to find out the right-most boundary by searching the the number equal to or greater then (d+k+1)*x where we will get k+1 multiples and we get the no of pairs as (right-left) boundary [index-wise].

```
 // cpp program to calculate the number
// pairs satisfying th condition
#include <bits/stdc++.h>
using namespace std;

// function to calculate the number of pairs
int countPairs(int a[], int n, int x, int k)
{
    sort(a, a + n);

    // traverse through all elements
    int ans = 0;
    for (int i = 0; i < n; i++) {

        // current number's divisor
        int d = (a[i] - 1) / x;

        // use binary search to find the element
        // after k multiples of x
        int it1 = lower_bound(a, a + n,
                        max((d + k) * x, a[i])) - a;

        // use binary search to find the element
        // after k+1 multiples of x so that we get
        // the answer bu subtracting
        int it2 = lower_bound(a, a + n,
                     max((d + k + 1) * x, a[i])) - a;

        // the difference of index will be the answer
        ans += it2 - it1;
    }
    return ans;
}

// driver code to check the above fucntion
int main()
```

```
{
    int a[] = { 1, 3, 5, 7 };
    int n = sizeof(a) / sizeof(a[0]);
    int x = 2, k = 1;

    // function call to get the number of pairs
    cout << countPairs(a, n, x, k);
    return 0;
}
```

Output:

```
3
```

Time complexity: O(n log n)

Source

https://www.geeksforgeeks.org/no-pairs-aj-ai-k-numbers-range-ai-aj-divisible-x/

Chapter 185

Non-Repeating Element

Non-Repeating Element - GeeksforGeeks

Find the first non-repeating element in a given array of integers.

Examples:

```
Input : -1 2 -1 3 2
Output : 3
Explanation : The first number that does not
repeat is : 3

Input : 9 4 9 6 7 4
Output : 6
```

A **Simple Solution** is to use two loops. The outer loop picks elements one by one and inner loop checks if the element is present more than once or not.

C++

```
 // Simple CPP program to find first non-
// repeating element.
#include <bits/stdc++.h>
using namespace std;

int firstNonRepeating(int arr[], int n)
{
    for (int i = 0; i < n; i++) {
        int j;
        for (j=0; j<n; j++)
            if (i != j && arr[i] == arr[j])
                break;
        if (j == n)
```

```
            return arr[i];
    }
    return -1;
}

// Driver code
int main()
{
    int arr[] = { 9, 4, 9, 6, 7, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << firstNonRepeating(arr, n);
    return 0;
}
```

Java

```
 // Java program to find first non-repeating
// element.
class GFG {

    static int firstNonRepeating(int arr[], int n)
    {
        for (int i = 0; i < n; i++) {
            int j;
            for (j = 0; j < n; j++)
                if (i != j && arr[i] == arr[j])
                    break;
            if (j == n)
                return arr[i];
        }

        return -1;
    }

    //Driver code
    public static void main (String[] args)
    {

        int arr[] = { 9, 4, 9, 6, 7, 4 };
        int n = arr.length;

        System.out.print(firstNonRepeating(arr, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find first
# non-repeating element.

def firstNonRepeating(arr, n):

    for i in range(n):
        j = 0
        while(j < n):
            if (i != j and arr[i] == arr[j]):
                break
            j += 1
        if (j == n):
            return arr[i]

    return -1

# Driver code
arr = [ 9, 4, 9, 6, 7, 4 ]
n = len(arr)
print(firstNonRepeating(arr, n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to find first non-
// repeating element.
using System;

class GFG
{
    static int firstNonRepeating(int []arr, int n)
    {
        for (int i = 0; i < n; i++) {
            int j;
            for (j = 0; j < n; j++)
                if (i != j && arr[i] == arr[j])
                    break;
            if (j == n)
                return arr[i];
        }
        return -1;
    }

    // Driver code
    public static void Main ()
    {
        int []arr = { 9, 4, 9, 6, 7, 4 };
```

```
        int n = arr.Length;
        Console.Write(firstNonRepeating(arr, n));
    }
}
// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// Simple PHP program to find first non-
// repeating element.

function firstNonRepeating($arr, $n)
{
    for ($i = 0; $i < $n; $i++)
    {
        $j;
        for ($j = 0; $j< $n; $j++)
            if ($i != $j && $arr[$i] == $arr[$j])
                break;
        if ($j == $n)
            return $arr[$i];
    }
    return -1;
}

    // Driver code
    $arr = array(9, 4, 9, 6, 7, 4);
    $n = sizeof($arr) ;
    echo firstNonRepeating($arr, $n);

// This code is contributed by ajit
?>
```

Output:

```
6
```

An **Efficient Solution** is to use hashing.
1) Traverse array and insert elements and their counts in hash table.
2) Traverse array again and print first element with count equals to 1.

```
 // Efficient CPP program to find first non-
// repeating element.
#include <bits/stdc++.h>
```

```
using namespace std;

int firstNonRepeating(int arr[], int n)
{
    // Insert all array elements in hash
    // table
    unordered_map<int, int> mp;
    for (int i = 0; i < n; i++)
       mp[arr[i]]++;

    // Traverse array again and return
    // first element with count 1.
    for (int i = 0; i < n; i++)
       if (mp[arr[i]] == 1)
            return arr[i];
    return -1;
}

// Driver code
int main()
{
    int arr[] = { 9, 4, 9, 6, 7, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << firstNonRepeating(arr, n);
    return 0;
}
```

Output:

```
6
```

Time Complexity : O(n)
Auxiliary Space : O(n)

Further Optimization: If array has many duplicates, we can also store index in hash table, using a hash table where value is a pair. Now we only need to traverse keys in hash table (not complete array) to find first non repeating.

Printing all non-repeating elements:

```
 // Efficient CPP program to print all non-
// repeating elements.
#include <bits/stdc++.h>
using namespace std;

void firstNonRepeating(int arr[], int n)
{
```

```
    // Insert all array elements in hash
    // table
    unordered_map<int, int> mp;
    for (int i = 0; i < n; i++)
       mp[arr[i]]++;

    // Traverse through map only and
    for (auto x : mp)
       if (x.second == 1)
            cout << x.first << " ";
}

// Driver code
int main()
{
    int arr[] = { 9, 4, 9, 6, 7, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    firstNonRepeating(arr, n);
    return 0;
}
```

Output:

```
7 6
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/non-repeating-element/

Chapter 186

Number of Larger Elements on right side in a string

Number of Larger Elements on right side in a string - GeeksforGeeks

Given a string, find count of number of larger alphabets for every character of the string.

Examples:

```
Input  : str = "abcd"
Output : 3 2 1 0
There are 3 greater characters on right of 'a',
2 greater characters on right of 'b', 1 greater
character on right of 'c' and 0 greater characters
on right of 'd'.

Input :  geeks
Output : 2 2 2 1 0
```

A **naive approach** is to use two for loops. First will keep track of each alphabet in string and second loop will be used to find no of larger alphabet according to ASCII values.

C++

```
 // CPP program to find counts of right greater
// characters for every character.
#include <bits/stdc++.h>
using namespace std;

void printGreaterCount(string str)
```

```
{
    int len = str.length(), right[len] = { 0 };
    for (int i = 0; i < len; i++)
        for (int j = i + 1; j < len; j++)
            if (str[i] < str[j])
                right[i]++;

    for (int i = 0; i < len; i++)
        cout << right[i] << " ";
}

// Driver code
int main()
{
    string str = "abcd";
    printGreaterCount(str);
    return 0;
}
```

PHP

```
 <?php
// PHP program to find counts
// of right greater characters
// for every character.

function printGreaterCount($str)
{
    $len = strlen($str);
    $right = array_fill(0, $len, 0);
    for ($i = 0; $i < $len; $i++)
    {
        for ($j = $i + 1; $j < $len; $j++)
            if ($str[$i] < $str[$j])
                $right[$i]++;
    }

    for ($i = 0; $i < $len; $i++)
        echo $right[$i] . " ";
}

// Driver code
$str = 'abcd';
printGreaterCount($str);

// This code is contributed
// by Abhinav96
?>
```

Output:

```
3 2 1 0
```

Time Complexity : O(N * N)

An **efficient approach** is to traverse the string from right and keep track of counts of characters from right side. For every character that we traverse from right, we increment its count in count array and add counts of all greater characters to answer for this character.

C++

```
 // C++ program to count greater characters on right
// side of every character.
#include <bits/stdc++.h>
using namespace std;
#define MAX_CHAR 26

void printGreaterCount(string str)
{
    int len = str.length();

    // Arrays to store result and character counts.
    int ans[len] = { 0 }, count[MAX_CHAR] = { 0 };

    // start from right side of string
    for (int i = len - 1; i >= 0; i--)
    {
        count[str[i] - 'a']++;
        for (int j = str[i] - 'a' + 1; j < MAX_CHAR; j++)
            ans[i] += count[j];
    }

    for (int i = 0; i < len; i++)
        cout << ans[i] << " ";
}

// Driver code
int main()
{
    string str = "abcd";
    printGreaterCount(str);
    return 0;
}
```

Output:

```
3 2 1 0
```

Time Complexity : O(N)

Source

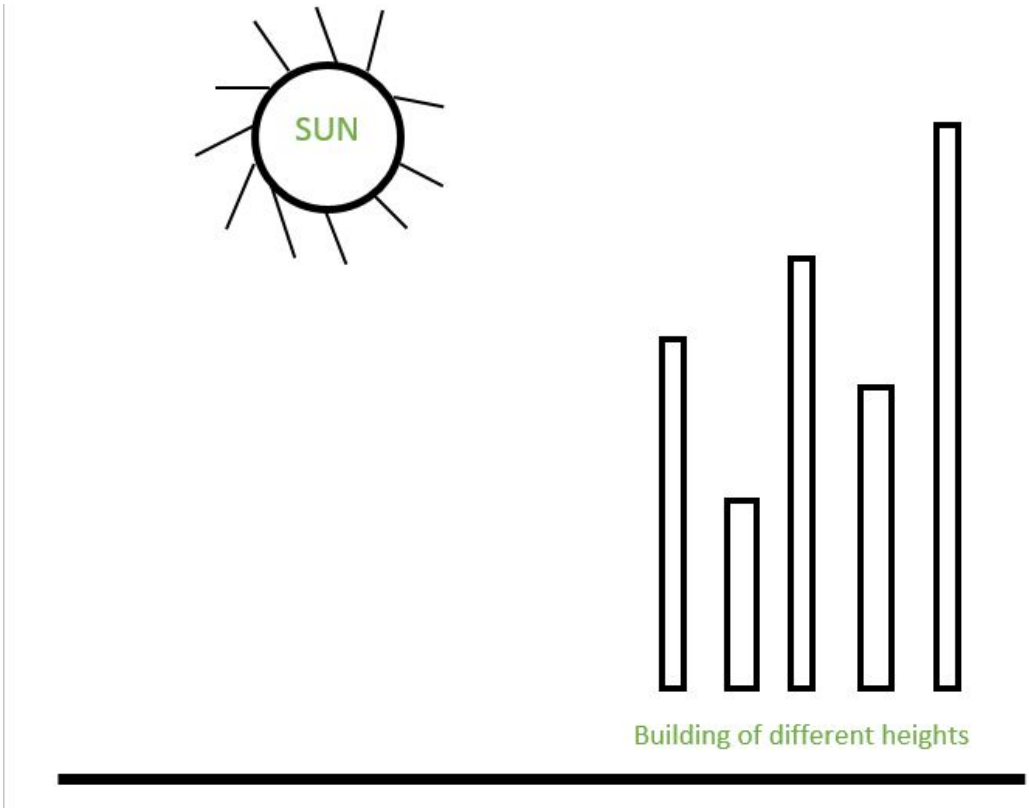
https://www.geeksforgeeks.org/number-of-larger-elements-on-right-side-in-a-string/

Chapter 187

Number of buildings facing the sun

Number of buildings facing the sun - GeeksforGeeks

Given an array representing heights of buildings. The array has buildings from left to right as shown in below diagram, count number of buildings facing the sunset. It is assumed that heights of all buildings are distinct.



Examples:

```
Input : arr[] = {7, 4, 8, 2, 9}
Output: 3
Explanation: As 7 is the first element, it can
see the sunset.
4 can't see the sunset as 7 is hiding it.
8 can see.
2 can't see the sunset.
9 also can see the sunset.

Input : arr[] = {2, 3, 4, 5}
Output : 4
```

Asked in : Amazon Interview

It can be easily observed that only the maximum element found so far will see the sunlight i.e. curr_max will see the sunlight and then only the element greater than curr_max will see the sunlight. We traverse given array from left to right. We keep track of maximum element seen so far. Whenever an element becomes more than current max, increment result and update current max.

C++

```
 // C++ program to count buildings that can
// see sunlight.
#include <iostream>
using namespace std;

// Returns count buildings that can see sunlight
int countBuildings(int arr[], int n)
{
    // Initialuze result  (Note that first building
    // always sees sunlight)
    int count = 1;

    // Start traversing element
    int curr_max = arr[0];
    for (int i=1; i<n; i++)
    {
        // If curr_element is maximum,
        // update maximum and increment count
        if (arr[i] > curr_max)
        {
            count++;
            curr_max=arr[i];
        }
    }

    return count;
}

// Driver code
int main()
{
    int arr[] = {7, 4, 8, 2, 9};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << countBuildings(arr, n);
    return 0;
}
```

Java

```
 // Java program to count buildings that can
// see sunlight.

class Test
{
    // Returns count buildings that can see sunlight
```

```
    static int countBuildings(int arr[], int n)
    {
        // Initialuze result  (Note that first building
        // always sees sunlight)
        int count = 1;

        // Start traversing element
        int curr_max = arr[0];
        for (int i=1; i<n; i++)
        {
            // If curr_element is maximum,
            // update maximum and increment count
            if (arr[i] > curr_max)
            {
                count++;
                curr_max=arr[i];
            }
        }

        return count;
    }

    // Driver method
    public static void main(String[] args)
    {
        int arr[] = {7, 4, 8, 2, 9};

        System.out.println(countBuildings(arr, arr.length));

    }
}
```

Python3

```
 # Python3 program to count buildings
# that can see sunlight.

# Returns count buildings that
# can see sunlight
def countBuildings(arr, n):

    # Initialuze result (Note that first
    # building always sees sunlight)
    count = 1

    # Start traversing element
    curr_max = arr[0]
    for i in range(1, n):
```

```
        # If curr_element is maximum,
        # update maximum and increment count
        if (arr[i] > curr_max):

            count += 1
            curr_max = arr[i]

    return count

# Driver code
arr = [7, 4, 8, 2, 9]
n = len(arr)
print(countBuildings(arr, n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# program to count buildings that can
// see sunlight.
using System;

class GFG
{
    // Returns count buildings that can see sunlight
    static int countBuildings(int []arr, int n)
    {
        // Initialuze result  (Note that first building
        // always sees sunlight)
        int count = 1;

        // Start traversing element
        int curr_max = arr[0];
        for (int i = 1; i < n; i++)
        {
            // If curr_element is maximum,
            // update maximum and increment count
            if (arr[i] > curr_max)
            {
                count++;
                curr_max = arr[i];
            }
        }

        return count;
    }
```

```
    // Driver method
    public static void Main()
    {
        int []arr = {7, 4, 8, 2, 9};

        Console.Write(countBuildings(arr, arr.Length));

    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
 <?php
// php program to count buildings
// that can see sunlight.

// Returns count buildings that
// can see sunlight
function countBuildings($arr, $n)
{
    // Initialuze result (Note that
    // first building always sees
    // sunlight)
    $count = 1;

    // Start traversing element
    $curr_max = $arr[0];
    for ( $i = 1; $i < $n; $i++)
    {
        // If curr_element is maximum,
        // update maximum and
        // increment count
        if ($arr[$i] > $curr_max)
        {
            $count++;
            $curr_max=$arr[$i];
        }
    }

    return $count;
}

// Driver code
$arr = array(7, 4, 8, 2, 9);
$n = sizeof($arr) / sizeof($arr[0]);
echo countBuildings($arr, $n);
```

```
// This code is contributed by
// nitin mittal
?>
```

Output:

```
3
```

Time Complexity : O(n)
Auxiliary Space : O(1)

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/number-buildings-facing-sun/

Chapter 188

Number of local extrema in an array

Number of local extrema in an array - GeeksforGeeks

You are given an array on n-elements. An extrema is an elements which is either greater than its both of neighbors or less than its both neighbors. You have to calculate the number of local extrema in given array.
Note : 1st and last elements are not extrema.

Examples :

```
Input : a[] = {1, 5, 2, 5}
Output : 2

Input : a[] = {1, 2, 3}
Output : 0
```

Approach : For calculating number of extrema we have to check whether an element is maxima or minima i.e. whether it is greater than both of its neighbors or less than both neighbors. For this simply iterate over the array and for each elements check its possibility of being an extrema.

Note: a[0] and a[n-1] has exactly one neighbour each, they are neither minima nor maxima.

C++

```
 // CPP to find number
// of extrema
#include <bits/stdc++.h>
using namespace std;

// function to find
```

```
// local extremum
int extrema(int a[], int n)
{
    int count = 0;

    // start loop from position 1
    // till n-1
    for (int i = 1; i < n - 1; i++)
    {

        // only one condition
        // will be true at a
        // time either a[i]
        // will be greater than
        // neighbours or less
        // than neighbours

        // check if a[i] is greater
        // than both its neighbours
        // then add 1 to x
        count += (a[i] > a[i - 1] && a[i] > a[i + 1]);

        // check if a[i] is
        // less than both its
        // neighbours, then
        // add 1 to x
        count += (a[i] < a[i - 1] && a[i] < a[i + 1]);
    }

    return count;
}

// driver program
int main()
{
    int a[] = { 1, 0, 2, 1 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << extrema(a, n);
    return 0;
}
```

Java

```
 // Java to find
// number of extrema
import java.io.*;

class GFG {
```

```
    // function to find
    // local extremum
    static int extrema(int a[], int n)
    {
        int count = 0;

        // start loop from
        // position 1 till n-1
        for (int i = 1; i < n - 1; i++)
        {

            // only one condition
            // will be true at a
            // time either a[i]
            // will be greater than
            // neighbours or less
            // than neighbours

            // check if a[i] is greater
            // than both its neighbours
            // then add 1 to x
            if(a[i] > a[i - 1] && a[i] > a[i + 1])
                count += 1;

            // check if a[i] is
            // less than both its
            // neighbours, then
            // add 1 to x
            if(a[i] < a[i - 1] && a[i] < a[i + 1])
                count += 1;
        }

        return count;
    }

    // driver program
    public static void main(String args[])
                            throws IOException
    {
        int a[] = { 1, 0, 2, 1 };
        int n = a.length;
        System.out.println(extrema(a, n));
    }
}


/* This code is contributed by Nikita Tiwari.*/
```

Python3

```
 # Python 3 to find
# number of extrema

# function to find
# local extremum
def extrema(a, n):

    count = 0
    # start loop from
    # position 1 till n-1
    for i in range(1, n - 1) :
        # only one condition
        # will be true
        # at a time either
        # a[i] will be greater
        # than neighbours or
        # less than neighbours

        # check if a[i] if
        # greater than both its
        # neighbours, then add
        # 1 to x
        count += (a[i] > a[i - 1] and a[i] > a[i + 1]);

        # check if a[i] if
        # less than both its
        # neighbours, then
        # add 1 to x
        count += (a[i] < a[i - 1] and a[i] < a[i + 1]);

    return count

# driver program
a = [1, 0, 2, 1 ]
n = len(a)
print(extrema(a, n))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# to find
// number of extrema
using System;
```

```
class GFG {

    // function to find
    // local extremum
    static int extrema(int []a, int n)
    {
        int count = 0;

        // start loop from
        // position 1 till n-1
        for (int i = 1; i < n - 1; i++)
        {

            // only one condition
            // will be true at a
            // time either a[i]
            // will be greater than
            // neighbours or less
            // than neighbours

            // check if a[i] is greater
            // than both its neighbours
            // then add 1 to x
            if(a[i] > a[i - 1] && a[i] > a[i + 1])
                count += 1;

            // check if a[i] is
            // less than both its
            // neighbours, then
            // add 1 to x
            if(a[i] < a[i - 1] && a[i] < a[i + 1])
                count += 1;
        }

        return count;
    }

    // Driver program
    public static void Main()

    {
        int []a = { 1, 0, 2, 1 };
        int n = a.Length;
    Console.WriteLine(extrema(a, n));
    }
}
```

```
/* This code is contributed by vt_m.*/
```

PHP

```
 <?php
// PHP to find number
// of extrema


// function to find
// local extremum
function extrema($a, $n)
{
    $count = 0;

    // start loop from position 1
    // till n-1
    for ($i = 1; $i < $n - 1; $i++)
    {

        // only one condition
        // will be true at a
        // time either a[i]
        // will be greater than
        // neighbours or less
        // than neighbours

        // check if a[i] is greater
        // than both its neighbours
        // then add 1 to x
        $count += ($a[$i] > $a[$i - 1] and
                   $a[$i] > $a[$i + 1]);

        // check if a[i] is
        // less than both its
        // neighbours, then
        // add 1 to x
        $count += ($a[$i] < $a[$i - 1] and
                   $a[$i] < $a[$i + 1]);
    }

    return $count;
}

// Driver Code
$a = array( 1, 0, 2, 1 );
$n = count($a);
echo extrema($a, $n);
```

```
// This code is contributed by anuj_67.
?>
```

Output :

```
2
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/maximum-number-local-extrema/

Chapter 189

Number of pairs with maximum sum

Number of pairs with maximum sum - GeeksforGeeks

Given an array arr[], count number of pairs arr[i], arr[j] such that arr[i] + arr[j] is maximum and i < j.

```
Example :
Input  : arr[] = {1, 1, 1, 2, 2, 2}
Output : 3
Explanation: The maximum possible pair
sum where i<j is  4, which is given
by 3 pairs, so the answer is 3
the pairs are (2, 2), (2, 2) and (2, 2)

Input  : arr[] = {1, 4, 3, 3, 5, 1}
Output : 1
Explanation: The pair 4, 5 yields the
maximum sum i.e, 9 which is given by 1 pair only
```

Method 1 (Naive)
Traverse a loop i from 0 to n, i.e length of the array and another loop j from i+1 to n to find all possible pairs with i<j. Find the pair with the maximum possible sum, again traverse for all pairs and keep the count of the number of pairs which gives the pair sum equal to maximum

C++

```
 // CPP program to count pairs with maximum sum.
#include <bits/stdc++.h>
```

```
using namespace std;

// function to find the number of maximum pair sums
int sum(int a[], int n)
{
    // traverse through all the pairs
    int maxSum = INT_MIN;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            maxSum = max(maxSum, a[i] + a[j]);

    // traverse through all pairs and keep a count
    // of the number of maximum pairs
    int c = 0;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (a[i] + a[j] == maxSum)
                c++;
    return c;
}

// driver program to test the above function
int main()
{
    int array[] = { 1, 1, 1, 2, 2, 2 };
    int n = sizeof(array) / sizeof(array[0]);
    cout << sum(array, n);
    return 0;
}
```

Java

```
 // Java program to count pairs
// with maximum sum.
class GFG {

// function to find the number of
// maximum pair sums
static int sum(int a[], int n)
{
    // traverse through all the pairs
    int maxSum = Integer.MIN_VALUE;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            maxSum = Math.max(maxSum, a[i] +
                                      a[j]);

    // traverse through all pairs and
```

```
    // keep a count of the number of
    // maximum pairs
    int c = 0;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (a[i] + a[j] == maxSum)
                c++;
    return c;
}

// driver program to test the above function
public static void main(String[] args)
{
    int array[] = { 1, 1, 1, 2, 2, 2 };
    int n = array.length;
    System.out.println(sum(array, n));
}
}

// This code is contributed by Prerna Saini
```

Python3

```
 # Python program to count pairs with
# maximum sum

def _sum( a, n):

    # traverse through all the pairs
    maxSum = -9999999
    for i in range(n):
        for j in range(n):
            maxSum = max(maxSum, a[i] + a[j])

    # traverse through all pairs and
    # keep a count of the number
    # of maximum pairs
    c = 0
    for i in range(n):
        for j in range(i+1, n):
            if a[i] + a[j] == maxSum:
                c+=1
    return c

# driver code
array = [ 1, 1, 1, 2, 2, 2 ]
n = len(array)
print(_sum(array, n))
```

```
# This code is contributed by "Abhishek Sharma 44"
```

C#

```
 // C# program to count pairs
// with maximum sum.
using System;

class GFG {

    // function to find the number of
    // maximum pair sums
    static int sum(int []a, int n)
    {

        // traverse through all the pairs
        int maxSum = int.MinValue;

        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                maxSum = Math.Max(maxSum,
                              a[i] + a[j]);

        // traverse through all pairs and
        // keep a count of the number of
        // maximum pairs
        int c = 0;
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                if (a[i] + a[j] == maxSum)
                    c++;
        return c;
    }

    // driver program to test the above
    // function
    public static void Main()
    {
        int []array = { 1, 1, 1, 2, 2, 2 };
        int n = array.Length;
        Console.WriteLine(sum(array, n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to count pairs
// with maximum sum.

// function to find the number
// of maximum pair sum
function sum( $a, $n)
{

    // traverse through all
    // the pairs
    $maxSum = PHP_INT_MIN;
    for($i = 0; $i < $n; $i++)
        for($j = $i + 1; $j < $n; $j++)
            $maxSum = max($maxSum, $a[$i] + $a[$j]);

    // traverse through all
    // pairs and keep a count
    // of the number of
    // maximum pairs
    $c = 0;
    for($i = 0; $i < $n; $i++)
        for($j = $i + 1; $j < $n; $j++)
            if ($a[$i] + $a[$j] == $maxSum)
                $c++;
    return $c;
}

    // Driver Code
    $array = array(1, 1, 1, 2, 2, 2);
    $n = count($array);
    echo sum($array, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
3
```

Time complexity:O(n^2)

Method 2 (Efficient)
If we take a closer look, we can notice following facts.

1. Maximum element is always part of solution

2. If maximum element appears more than once, then result is maxCount * (maxCount – 1)/2. We basically need to choose 2 elements from maxCount (${}^{\text{maxCount}}\text{C}_2$).
3. If maximum element appears once, then result is equal to count of second maximum element. We can form a pair with every second max and max

C++

```
 // CPP program to count pairs with maximum sum.
#include <bits/stdc++.h>
using namespace std;

// function to find the number of maximum pair sums
int sum(int a[], int n)
{
    // Find maximum and second maximum elements.
    // Also find their counts.
    int maxVal = a[0], maxCount = 1;
    int secondMax = INT_MIN, secondMaxCount;
    for (int i = 1; i < n; i++) {
        if (a[i] == maxVal)
            maxCount++;
        else if (a[i] > maxVal) {
            secondMax = maxVal;
            secondMaxCount = maxCount;
            maxVal = a[i];
            maxCount = 1;
        }
        else if (a[i] == secondMax) {
            secondMax = a[i];
            secondMaxCount++;
        }
        else if (a[i] > secondMax) {
            secondMax = a[i];
            secondMaxCount = 1;
        }
    }

    // If maximum element appears more than once.
    if (maxCount > 1)
        return maxCount * (maxCount - 1) / 2;

    // If maximum element appears only once.
    return secondMaxCount;
}

// driver program to test the above function
int main()
{
```

```
    int array[] = { 1, 1, 1, 2, 2, 2, 3 };
    int n = sizeof(array) / sizeof(array[0]);
    cout << sum(array, n);
    return 0;
}
```

Java

```
 // Java program to count pairs
// with maximum sum.
import java.io.*;
class GFG {

// function to find the number
// of maximum pair sums
static int sum(int a[], int n)
{
    // Find maximum and second maximum
    // elements. Also find their counts.
    int maxVal = a[0], maxCount = 1;
    int secondMax = Integer.MIN_VALUE,
        secondMaxCount = 0;
    for (int i = 1; i < n; i++) {
        if (a[i] == maxVal)
            maxCount++;
        else if (a[i] > maxVal) {
            secondMax = maxVal;
            secondMaxCount = maxCount;
            maxVal = a[i];
            maxCount = 1;
        }
        else if (a[i] == secondMax) {
            secondMax = a[i];
            secondMaxCount++;
        }
        else if (a[i] > secondMax) {
            secondMax = a[i];
            secondMaxCount = 1;
        }
    }

    // If maximum element appears
    // more than once.
    if (maxCount > 1)
        return maxCount * (maxCount - 1) / 2;

    // If maximum element appears
    // only once.
```

```
    return secondMaxCount;
}

// driver program
public static void main(String[] args)
{
    int array[] = { 1, 1, 1, 2, 2, 2, 3 };
    int n = array.length;
    System.out.println(sum(array, n));
}
}

// This code is contributed by Prerna Saini
```

Python3

```
 # Python 3 program to count
# pairs with maximum sum.
import sys

# Function to find the number
# of maximum pair sums
def sum(a, n):

    # Find maximum and second maximum elements.
    # Also find their counts.
    maxVal = a[0]; maxCount = 1
    secondMax = sys.maxsize

    for i in range(1, n) :

        if (a[i] == maxVal) :
            maxCount += 1

        elif (a[i] > maxVal) :
            secondMax = maxVal
            secondMaxCount = maxCount
            maxVal = a[i]
            maxCount = 1

        elif (a[i] == secondMax) :
            secondMax = a[i]
            secondMaxCount += 1

        elif (a[i] > secondMax) :
            secondMax = a[i]
            secondMaxCount = 1
```

```
    # If maximum element appears more than once.
    if (maxCount > 1):
        return maxCount * (maxCount - 1) / 2

    # If maximum element appears only once.
    return secondMaxCount

# Driver Code
array = [1, 1, 1, 2, 2, 2, 3]
n = len(array)
print(sum(array, n))


# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to count pairs with maximum
// sum.
using System;

class GFG {

    // function to find the number
    // of maximum pair sums
    static int sum(int []a, int n)
    {

        // Find maximum and second maximum
        // elements. Also find their counts.
        int maxVal = a[0], maxCount = 1;
        int secondMax = int.MinValue;
        int secondMaxCount = 0;
        for (int i = 1; i < n; i++)
        {
            if (a[i] == maxVal)
                maxCount++;
            else if (a[i] > maxVal)
            {
                secondMax = maxVal;
                secondMaxCount = maxCount;
                maxVal = a[i];
                maxCount = 1;
            }
            else if (a[i] == secondMax)
            {
                secondMax = a[i];
                secondMaxCount++;
```

```
            }
            else if (a[i] > secondMax)
            {
                secondMax = a[i];
                secondMaxCount = 1;
            }
        }

        // If maximum element appears
        // more than once.
        if (maxCount > 1)
            return maxCount *
                        (maxCount - 1) / 2;

        // If maximum element appears
        // only once.
        return secondMaxCount;
    }

    // driver program
    public static void Main()
    {
        int []array = { 1, 1, 1, 2,
                                2, 2, 3 };
        int n = array.Length;

        Console.WriteLine(sum(array, n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to count
// pairs with maximum sum.

// function to find the number
// of maximum pair sums
function sum( $a, $n)
{
    // Find maximum and second
    // maximum elements. Also
    // find their counts.
    $maxVal = $a[0]; $maxCount = 1;
    $secondMax = PHP_INT_MIN;
    $secondMaxCount;
```

```
    for ( $i = 1; $i < $n; $i++)
    {
        if ($a[$i] == $maxVal)
            $maxCount++;
        else if ($a[$i] > $maxVal)
        {
            $secondMax = $maxVal;
            $secondMaxCount = $maxCount;
            $maxVal = $a[$i];
            $maxCount = 1;
        }
        else if ($a[$i] == $secondMax)
        {
            $secondMax = $a[$i];
            $secondMaxCount++;
        }
        else if ($a[$i] > $secondMax)
        {
            $secondMax = $a[$i];
            $secondMaxCount = 1;
        }
    }

    // If maximum element appears
    // more than once.
    if ($maxCount > 1)
        return $maxCount *
              ($maxCount - 1) / 2;

    // If maximum element
    // appears only once.
    return $secondMaxCount;
}

// Driver Code
$array = array(1, 1, 1, 2,
               2, 2, 3 );
$n = count($array);
echo sum($array, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
3
```

Time complexity:O(n)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/number-pairs-maximum-sum/

Chapter 190

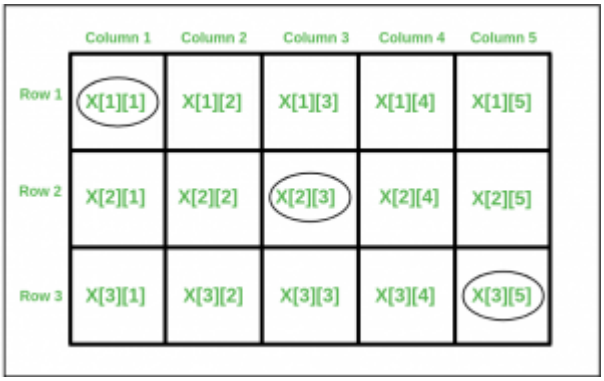
Number of positions with Same address in row major and column major order

Number of positions with Same address in row major and column major order - GeeksforGeeks

Given a 2D array of size M x N. Calculate count of positions in 2D array where address as per row major order equals to address as per column major order.

Examples:

Input : 3 5
Output : 3
Row major address is same as column major for following i, j
pairs (1, 1), (2, 3) & (3, 5)



Input : 4 4
Output : 4

Lets consider element with index i, j

```
Row major address = B + w * (N * (i-1) + j-1)
Column major address = B + w * (M * (j-1) + i-1)
```

B : Base address of the array
w : Size of each element of the array

```
Equating both addresses, we get
B + w * (N * (i-1) + j-1)  = B + w * (M * (j-1) + i-1)
N * (i-1) + j  = M * (j-1) + i
N*i - N + j  = M*j - M + i
M*j - j = N*i - N + M - i
(M-1) * j = N*i - N + M - i
j = (N*i - N + M - i)/(M-1)    - (Eq. 1)
Similarly
i = (M*j - M + N - j)/(N-1)    - (Eq. 2)
```

Now we have established a relation between i and j

```
Iterate for all possible i and find corresponding j
If j comes out to be an integer in the range 1 to N,
increment the counter.
```

Below is the implementation of above approach.

C++

```
 // CPP Program to count the number
// of positions with same address
// in row major and column major order
#include <bits/stdc++.h>

using namespace std;

// Returns count of required positions
int getCount(int M, int N)
{
    int count = 0;

    // horizontal 1D array
    if (M == 1)
        return N;
```

```
    // vertical 1D array
    if (N == 1)
        return M;

    if (N > M) {

        // iterating for all possible i
        for (int i = 1; i <= M; i++) {
            int numerator = N * i - N + M - i;
            int denominator = M - 1;

            // checking if j is integer
            if (numerator % denominator == 0) {
                int j = numerator / denominator;

                // checking if j lies b/w 1 to N
                if (j >= 1 && j <= N)
                    count++;
            }
        }
    }
    else {

        // iterating for all possible j
        for (int j = 1; j <= N; j++) {
            int numerator = M * j - M + N - j;
            int denominator = N - 1;

            // checking if i is integer
            if (numerator % denominator == 0) {
                int i = numerator / denominator;

                // checking if i lies b/w 1 to M
                if (i >= 1 && i <= M)
                    count++;
            }
        }
    }
    return count;
}

// Driver Code
int main()
{
    int M = 3, N = 5;
    cout << getCount(M, N) << endl;
    return 0;
```

```
}
```

Java

```
 // Java Program to count the number
// of positions with same address
// in row major and column major order
import java.io.*;
class GFG {

// Returns count of
// required positions
static int getCount(int M, int N)
{
    int count = 0;

    // horizontal 1D array
    if (M == 1)
        return N;

    // vertical 1D array
    if (N == 1)
        return M;

    if (N > M) {

        // iterating for all possible i
        for (int i = 1; i <= M; i++) {
            int numerator = N * i - N + M - i;
            int denominator = M - 1;

            // checking if j is integer
            if (numerator % denominator == 0) {
                int j = numerator / denominator;

                // checking if j lies b/w 1 to N
                if (j >= 1 && j <= N)
                    count++;
            }
        }
    }
    else {

        // iterating for all possible j
        for (int j = 1; j <= N; j++) {
            int numerator = M * j - M + N - j;
            int denominator = N - 1;
```

```
            // checking if i is integer
            if (numerator % denominator == 0) {
                int i = numerator / denominator;

                // checking if i lies b/w 1 to M
                if (i >= 1 && i <= M)
                    count++;
            }
        }
    }
    return count;
}

    // Driver Code
    public static void main (String[] args)
    {
        int M = 3, N = 5;
        System.out.println( getCount(M, N));
    }
}

// This code is contributed by vt_m.
```

C#

```
 // C# Program to count the number
// of positions with same address
// in row major and column major order
using System;
class GFG {

    // Returns count of
    // required positions
    static int getCount(int M, int N)
    {
        int count = 0;

        // horizontal 1D array
        if (M == 1)
            return N;

        // vertical 1D array
        if (N == 1)
            return M;

        if (N > M) {

            // iterating for all possible i
```

```
            for (int i = 1; i <= M; i++) {
                int numerator = N * i - N + M - i;
                int denominator = M - 1;

                // checking if j is integer
                if (numerator % denominator == 0) {
                    int j = numerator / denominator;

                    // checking if j lies b/w 1 to N
                    if (j >= 1 && j <= N)
                        count++;
                }
            }
        }
        else {

            // iterating for all possible j
            for (int j = 1; j <= N; j++) {
                int numerator = M * j - M + N - j;
                int denominator = N - 1;

                // checking if i is integer
                if (numerator % denominator == 0) {
                    int i = numerator / denominator;

                    // checking if i lies b/w 1 to M
                    if (i >= 1 && i <= M)
                        count++;
                }
            }
        }
        return count;
    }

    // Driver Code
    public static void Main ()
    {
        int M = 3, N = 5;
        Console.WriteLine( getCount(M, N));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP Program to count the number
```

```
// of positions with same address
// in row major and column major order

// Returns count of required positions
function getCount( $M, $N)
{
    $count = 0;

    // horizontal 1D array
    if ($M == 1)
        return $N;

    // vertical 1D array
    if ($N == 1)
        return $M;

    if ($N > $M)
    {

        // iterating for all possible i
        for($i = 1; $i <= $M; $i++)
        {
            $numerator = $N * $i - $N + $M - $i;
            $denominator = $M - 1;

            // checking if j is integer
            if ($numerator % $denominator == 0)
            {
                $j = $numerator / $denominator;

                // checking if j lies b/w 1 to N
                if ($j >= 1 and $j <= $N)
                    $count++;
            }
        }
    }
    else
    {

        // iterating for all possible j
        for ( $j = 1; $j <= $N; $j++)
        {
            $numerator = $M * $j - $M + $N - $j;
            $denominator = $N - 1;

            // checking if i is integer
            if ($numerator % $denominator == 0)
            {
```

```
                $i = $numerator / $denominator;

                // checking if i lies b/w 1 to M
                if ($i >= 1 and $i <= $M)
                    $count++;
            }
        }
    }
    return $count;
}

    // Driver Code
    $M = 3; $N = 5;
    echo getCount($M, $N) ;

// This code is contributed by anuj_67.
?>
```

Output :

```
3
```

Time Complexity: O(M)
Complexity can be reduced to O(min(M, N)) by establishing relation of i in terms of j(Eq. 2) and iterating for all possible j in case N<M and by establishing relation j in terms of i(Eq. 1) and iterating for all possible i otherwise.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/number-positions-address-row-major-column-major-order/

Chapter 191

Number of unique triplets whose XOR is zero

Number of unique triplets whose XOR is zero - GeeksforGeeks

Given N numbers with no duplicates, count the number of unique triplets $(\mathbf{a_i}, \mathbf{a_j}, \mathbf{a_k})$ such that their XOR is 0. A triplet is said to be unique if all of the three numbers in the triplet is unique.

Examples:

```
Input : a[] = {1, 3, 5, 10, 14, 15};
Output : 2
Explanation : {1, 14, 15} and {5, 10, 15} are the
              unique triplets whose XOR is 0.
              {1, 14, 15} and all other combinations of
              1, 14, 15 are considered as 1 only.

Input : a[] = {4, 7, 5, 8, 3, 9};
Output : 1
Explanation : {4, 7, 3} is the only triplet whose XOR is 0
```

Naive Approach: A naive approach is to run three nested loops, the first runs from 0 to n, second from i+1 to n, and the last one from j+1 to n to get the unique triplets. Calculate the XOR of a_i, a_j, a_k, check if it equals to 0, if so, then increase the count.
Time Complexity : $O(n^3)$

Efficient Approach: An efficient approach is to use one of the properties of XOR that XOR of two same numbers gives 0. So we need to calculate XOR of unique pairs only, and if the calculated XOR is one of the array element, then we get the triplet whose XOR is 0. Given below are the steps for counting the number of unique triplets:

Below is the complete algorithm of this approach:

1. With map, mark all the array elements.
2. Run two nested loops, one from i-n, and the other from i+1-n to get all the pairs.
3. Obtain the XOR of pair.
4. Check if the XOR is an array element and not one of a_i or a_j.
5. Increase the count if the condition holds.
6. Return count/3 as we only want unique triplets. Since i-n and j+1-n gives us unique pairs but not triplets, so we do a count/3 to remove the other two possible combinations.

Below is the implementation of above idea:

C++

```
 // CPP program to count the number of
// unique triplets whose XOR is 0
#include <bits/stdc++.h>
using namespace std;

// function to count the number of
// unique triplets whose xor is 0
int countTriplets(int a[], int n)
{
    // To store values that are present
    unordered_set<int> s;
    for (int i = 0; i < n; i++)
        s.insert(a[i]);

    // stores the count of unique triplets
    int count = 0;

    // traverse for all i, j pairs such that j>i
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {

          // xor of a[i] and a[j]
          int xr = a[i] ^ a[j];

          // if xr of two numbers is present,
          // then increase the count
          if (s.find(xr) != s.end() && xr != a[i] &&
                                       xr != a[j])
            count++;
        }
    }

    // returns answer
    return count / 3;
```

```
}

// Driver code to test above function
int main()
{
    int a[] = {1, 3, 5, 10, 14, 15};
    int n = sizeof(a) / sizeof(a[0]);
    cout << countTriplets(a, n);
    return 0;
}
```

Java

```
 // Java program to count
// the number of unique
// triplets whose XOR is 0
import java.io.*;
import java.util.*;

class GFG
{
    // function to count the
    // number of unique triplets
    // whose xor is 0
    static int countTriplets(int []a,
                             int n)
    {
        // To store values
        // that are present
        ArrayList<Integer> s =
                  new ArrayList<Integer>();
        for (int i = 0; i < n; i++)
            s.add(a[i]);

        // stores the count
        // of unique triplets
        int count = 0;

        // traverse for all i,
        // j pairs such that j>i
        for (int i = 0; i < n; i++)
        {
            for (int j = i + 1;
                     j < n; j++)
            {

            // xor of a[i] and a[j]
            int xr = a[i] ^ a[j];
```

```
            // if xr of two numbers
            // is present, then
            // increase the count
            if (s.contains(xr) &&
                xr != a[i] && xr != a[j])
                count++;
            }
        }

        // returns answer
        return count / 3;
    }

    // Driver code
    public static void main(String srgs[])
    {
        int []a = {1, 3, 5,
                   10, 14, 15};
        int n = a.length;
        System.out.print(countTriplets(a, n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
 // C# program to count
// the number of unique
// triplets whose XOR is 0
using System;
using System.Collections.Generic;

class GFG
{
    // function to count the
    // number of unique triplets
    // whose xor is 0
    static int countTriplets(int []a,
                             int n)
    {
        // To store values
        // that are present
        List<int> s = new List<int>();
        for (int i = 0; i < n; i++)
            s.Add(a[i]);
```

```
        // stores the count
        // of unique triplets
        int count = 0;

        // traverse for all i,
        // j pairs such that j>i
        for (int i = 0; i < n; i++)
        {
            for (int j = i + 1;
                     j < n; j++)
            {

            // xor of a[i] and a[j]
            int xr = a[i] ^ a[j];

            // if xr of two numbers
            // is present, then
            // increase the count
            if (s.Exists(item => item == xr) &&
                   xr != a[i] && xr != a[j])
                count++;
            }
        }

        // returns answer
        return count / 3;
    }

    // Driver code
    static void Main()
    {
        int []a = new int[]{1, 3, 5,
                            10, 14, 15};
        int n = a.Length;
        Console.Write(countTriplets(a, n));
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

```
2
```

Time Complexity : $O(n^2)$

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/number-unique-triplets-whose-xor-zero/

Chapter 192

Numbers within a range that can be expressed as power of two numbers

Numbers within a range that can be expressed as power of two numbers - GeeksforGeeks

Given two integers L and R. Find the number of perfect powersin the given range [L, R]. A number x is said to be perfect power if there exists some integers a > 0, p > 1 such that x = a^p.

Examples :

```
Input : 1 4
Output : 2
Explanation :
Suitable numbers are 1 and 4 where 1
can be expressed as 1 = 12
and 4 can be expressed as 4 = 22

Input : 12 29
Output : 3
Explanation :
Suitable numbers are 16, 25 and 27.
```

Prerequisites : Check if a number can be expressed as x^y, Binary Search and Perfect power (1, 4, 8, 9, 16, 25, 27, ...)

Approach : Let's fix some power p. It's obvious that there are no more than $10^{18/p}$ numbers x such that x^p doesn't exceed 10^{18} for a particular p. At the same time, only for p = 2 this amount is relatively huge, for all other p 3 the total amount of such numbers

will be of the order of 10^6. There are 10^9 squares in the range [1, 10^{18}], so can't store them to answer our query.

Either, generate all of powers for p 2 and dispose of all perfect squares among them or generate only odd powers of numbers like 3, 5, 7, etc. Then answer to query (L, R) is equal to the amount of generated numbers between L and R plus some perfect squares in range.

1. The number of perfect squares in the range is the difference of floor value of square root of R and floor value of square root of (L – 1), i.e. (**floor(sqrt(R)) − floor(sqrt(L − 1)**). *Note that due to precision issues the standard sqrt might produce incorrect values, so either use binary search or* ***sqrtl*** *inbuilt function defined in cmath (Check here for more description of sqrtl).*
2. To generate those odd powers of numbers. First of all, do precomputation of finding such numbers that can be expressed as power of some number upto 10^{18} so that we can answer many queries and no need to process them again and again for each query. Start by iterating a loop from 2 to 10^6 (since we are calculating for powers p 3 and 10^6 is the maximum number whose power raised to 3 cannot exceed 10^{18}), for each value we insert its square into a set and check further if that value is already a perfect square (already present in the set), we do not find any other powers of that number (since any power of a perfect square is also a perfect square). Otherwise, run an inside loop to find odd powers of the number until it exceeds 10^{18} and insert into another set say 's'. By this approach, we haven't pushed any perfect square in the set 's'.

Hence the final answer would be sum of number of perfect squares in the range and difference of upper value of R and lower value of L (using binary search).
Below is the implementation of above approach in C++.

```
 // CPP Program to count the numbers
// within a range such that number
// can be expressed as power of some
// other number
#include <bits/stdc++.h>

using namespace std;

#define N 1000005
#define MAX 1e18

// Vector to store powers greater than 3
vector<long int> powers;

// set to store perfect squares
set<long int> squares;

// set to store powers other
// than perfect squares
set<long int> s;
```

```
void powersPrecomputation()
{
    for (long int i = 2; i < N; i++)
    {
        // pushing squares
        squares.insert(i * i);

        // if the values is already
        // a perfect square means
        // present in the set
        if (squares.find(i) != squares.end())
                continue;

        long int temp = i;

        // run loop until some
        // power of current number
        // doesn't exceed MAX
        while (i * i <= MAX / temp)
        {
            temp *= (i * i);

            /* pushing only odd powers
            as even power of a number
            can always be expressed as
            a perfect square which is
            already present in set squares  */
            s.insert(temp);
        }
    }

    // Inserting those sorted
    // values of set into a vector
    for (auto x : s)
        powers.push_back(x);
}

long int calculateAnswer(long int L, long int R)
{
    // calculate perfect squares in
    // range using sqrtl function
    long int perfectSquares = floor(sqrtl(R)) -
                              floor(sqrtl(L - 1));

    // calculate upper value of R
    // in vector using binary search
    long int high = (upper_bound(powers.begin(),
            powers.end(), R) - powers.begin());
```

```
    // calculate lower value of L
    // in vector using binary search
    long int low = (lower_bound(powers.begin(),
            powers.end(), L) - powers.begin());

    // add into final answer
    perfectSquares += (high - low);

    return perfectSquares;
}

// Driver Code
int main()
{
    // precompute the powers
    powersPrecomputation();

    // left value of range
    long int L = 12;

    // right value of range
    long int R = 29;

    cout << "Number of powers between " << L
            << " and " << R << " = " <<
            calculateAnswer(L, R) << endl;

    L = 1;
    R = 100000000000;

    cout << "Number of powers between " << L
            << " and " << R << " = " <<
            calculateAnswer(L, R) << endl;

    return 0;
}
```

Output:

```
Number of powers between 12 and 29 = 3
Number of powers between 1 and 100000000000 = 320990
```

Source

https://www.geeksforgeeks.org/numbers-within-range-can-expressed-power-two-numbers/

Chapter 193

Pair with given sum and maximum shortest distance from end

Pair with given sum and maximum shortest distance from end - GeeksforGeeks

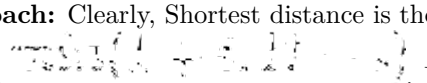
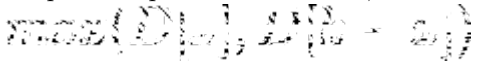
Given an array of N integers and an integer K, pick two distinct elements whose sum is K and find the maximum shortest distance of the picked elements from the endpoints.

Examples:

```
Input : a[] = {2, 4, 3, 2, 1}
        k = 5.
Output :  2
Explanation:
Select the pair(4, 1).
Shortest distance of 4 from ends = 2
Shortest distance of 1 from ends = 1
Hence, answer is max(2, 1) = 2

Input : a[] = {2, 4, 1, 9, 5}
        k = 3
Output : 3
Explanation:
Select the pair (2, 1)
Shortest distance of 2 from ends = 1
Shortest distance of 1 from ends = 3
Hence, answer is max(1, 3) = 3.
```

Note: The distance of end elements from ends is 1 and not 0.

Naive approach: The approach is to run two loops and in inner loop check if two elements are making a pair with sum k. If yes, then make answer as maximum of the shortest distances of two elements, compare it with the previous pair's answer and make answer as minimum of these two. When the loop ends we get the desired output.

Efficient Approach: Clearly, Shortest distance is the distance from left end and distance from right end i.e,  . Let us denote shortest distance of i-th element as . There is another case where an element in the selected pair is repeated then select minimum of all the shortest distances of occurrences of that element. Run a loop and store shortest distance of all the array elements in another array(let it be ). Now, we got shortest distances of all the elements.
Run a for loop. If the picked element is **x**, then the other element should be **k-x**. Update the ans with  and at every update, select the minimum of previous and present answer. If **k-x** is not in the array, then will be INFINITE, which will be initialized already.

```
 // C++ code to find maximum shortest distance
// from endpoints
#include <bits/stdc++.h>
using namespace std;

// function to find maximum shortest distance
int find_maximum(int a[], int n, int k)
{
    // stores the shortest distance of every
    // element in original array.
    unordered_map<int, int> b;

    for (int i = 0; i < n; i++) {
        int x = a[i];

        // shortest distance from ends
        int d = min(1 + i, n - i);
        if (b.find(x) == b.end())
            b[x] = d;

        else

            /* if duplicates are found, b[x]
            is replaced with minimum of the
            previous and current position's
            shortest distance*/
            b[x] = min(d, b[x]);
    }
```

```
    int ans = INT_MAX;
    for (int i = 0; i < n; i++) {
        int x = a[i];

        // similar elements ignore them
        // cause we need distinct elements
        if (x != k - x && b.find(k - x) != b.end())
            ans = min(max(b[x], b[k - x]), ans);
    }
    return ans;
}

// driver code
int main()
{
    int a[] = { 3, 5, 8, 6, 7 };
    int K = 11;
    int n = sizeof(a) / sizeof(a[0]);
    cout << find_maximum(a, n, K) << endl;
    return 0;
}
```

Output:

```
 2
```

Source

https://www.geeksforgeeks.org/print-maximum-shortest-distance/

Chapter 194

Pairs such that one is a power multiple of other

Pairs such that one is a power multiple of other - GeeksforGeeks

You are given an array A[] of n-elements and a positive integer k. Now you have find the number of pairs Ai, Aj such that $\mathbf{Ai = Aj*(k^x)}$ where x is an integer.
Note: (Ai, Aj) and (Aj, Ai) must be count once.

Examples :

```
Input : A[] = {3, 6, 4, 2},  k = 2
Output : 2
Explanation : We have only two pairs
(4, 2) and (3, 6)

Input : A[] = {2, 2, 2},   k = 2
Output : 3
Explanation : (2, 2), (2, 2), (2, 2)
that are (A1, A2), (A2, A3) and (A1, A3) are
total three pairs where Ai = Aj * (k^0)
```

To solve this problem, we first sort the given array and then for each element Ai, we find number of elements equal to value Ai * k^x for different value of x till Ai * k^x is less than or equal to largest of Ai.
Algorithm:

```
    // sort the given array
    sort(A, A+n);

    // for each A[i] traverse rest array
```

```
    for (int i=0; i<n; i++)
    {
        for (int j=i+1; j<n; j++)
        {
            // count Aj such that Ai*k^x = Aj
            int x = 0;

            // increase x till Ai * k^x <=
            // largest element
            while ((A[i]*pow(k, x)) <= A[j])
            {
                if ((A[i]*pow(k, x)) == A[j])
                {
                     ans++;
                     break;
                }
                x++;
            }
        }
    }
    // return answer
    return ans;
```

C++

```
 // Program to find pairs count
#include <bits/stdc++.h>
using namespace std;

// function to count the required pairs
int countPairs(int A[], int n, int k) {
  int ans = 0;
  // sort the given array
  sort(A, A + n);

  // for each A[i] traverse rest array
  for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {

      // count Aj such that Ai*k^x = Aj
      int x = 0;

      // increase x till Ai * k^x <= largest element
      while ((A[i] * pow(k, x)) <= A[j]) {
        if ((A[i] * pow(k, x)) == A[j]) {
          ans++;
          break;
        }
```

```
        x++;
      }
    }
  }
  return ans;
}

// driver program
int main() {
  int A[] = {3, 8, 9, 12, 18, 4, 24, 2, 6};
  int n = sizeof(A) / sizeof(A[0]);
  int k = 3;
  cout << countPairs(A, n, k);
  return 0;
}
```

Java

```
 // Java program to find pairs count
import java.io.*;
import java .util.*;

class GFG {

    // function to count the required pairs
    static int countPairs(int A[], int n, int k)
    {
        int ans = 0;

        // sort the given array
        Arrays.sort(A);

        // for each A[i] traverse rest array
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++)
            {

                // count Aj such that Ai*k^x = Aj
                int x = 0;

                // increase x till Ai * k^x <= largest element
                while ((A[i] * Math.pow(k, x)) <= A[j])
                {
                    if ((A[i] * Math.pow(k, x)) == A[j])
                    {
                        ans++;
                        break;
                    }
```

```
                    x++;
                }
            }
        }
        return ans;
    }

    // Driver program
    public static void main (String[] args)
    {
        int A[] = {3, 8, 9, 12, 18, 4, 24, 2, 6};
        int n = A.length;
        int k = 3;
        System.out.println (countPairs(A, n, k));

    }
}

// This code is contributed by vt_m.
```

Python3

```
 # Program to find pairs count
import math

# function to count the required pairs
def countPairs(A, n, k):
    ans = 0

    # sort the given array
    A.sort()

    # for each A[i] traverse rest array
    for i in range(0,n):

        for j in range(i + 1, n):

            # count Aj such that Ai*k^x = Aj
            x = 0

            # increase x till Ai * k^x <= largest element
            while ((A[i] * math.pow(k, x)) <= A[j]) :
                if ((A[i] * math.pow(k, x)) == A[j]) :
                    ans+=1
                    break
                x+=1
    return ans
```

```
# driver program
A = [3, 8, 9, 12, 18, 4, 24, 2, 6]
n = len(A)
k = 3

print(countPairs(A, n, k))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to find pairs count
using System;

class GFG {

    // function to count the required pairs
    static int countPairs(int []A, int n, int k)
    {
        int ans = 0;

        // sort the given array
        Array.Sort(A);

        // for each A[i] traverse rest array
        for (int i = 0; i < n; i++)
        {
            for (int j = i + 1; j < n; j++)
            {

                // count Aj such that Ai*k^x = Aj
                int x = 0;

                // increase x till Ai * k^x <= largest element
                while ((A[i] * Math.Pow(k, x)) <= A[j])
                {
                    if ((A[i] * Math.Pow(k, x)) == A[j])
                    {
                        ans++;
                        break;
                    }
                    x++;
                }
            }
        }
        return ans;
```

```
    }

    // Driver program
    public static void Main ()
    {
        int []A = {3, 8, 9, 12, 18, 4, 24, 2, 6};
        int n = A.Length;
        int k = 3;
        Console.WriteLine(countPairs(A, n, k));

    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to find pairs count

// function to count
// the required pairs
function countPairs($A, $n, $k)
{
$ans = 0;

// sort the given array
sort($A);

// for each A[i]
// traverse rest array
for ($i = 0; $i < $n; $i++)
{
    for ($j = $i + 1; $j < $n; $j++)
    {

    // count Aj such that Ai*k^x = Aj
    $x = 0;

    // increase x till Ai *
    // k^x <= largest element
    while (($A[$i] * pow($k, $x)) <= $A[$j])
    {
        if (($A[$i] * pow($k, $x)) == $A[$j])
        {
        $ans++;
        break;
        }
```

```
        $x++;
    }
    }
}
return $ans;
}

// Driver Code

$A = array(3, 8, 9, 12, 18,
              4, 24, 2, 6);
$n = count($A);
$k = 3;
echo countPairs($A, $n, $k);

// This code is contributed by anuj_67.
?>
```

Output :

```
6
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/pairs-one-power-multiple/

Chapter 195

Path in a Rectangle with Circles

Path in a Rectangle with Circles - GeeksforGeeks

There is a m*n rectangular matrix whose top-left(start) location is (1, 1) and bottom-right(end) location is (m*n). There are k circles each with radius r. Find if there is any path from start to end without touching any circle.

The input contains values of m, n, k, r and two array of integers X and Y, each of length k. (X[i], Y[i]) is the centre of ith circle.

Source : Directi Interview

```
Input1 : m = 5, n = 5, k = 2, r = 1,
         X = {1, 3}, Y = {3, 3}
Output1 : Possible

Here is a path from start to end point.

Input2 : m = 5, n = 5, k = 2, r = 1,
         X = {1, 1}, Y = {2, 3}.
Output2 : Not Possible
```

Approach : Check if the centre of a cell (i, j) of the rectangle comes within any of the circles then do not traverse through that cell and mark that as 'blocked'. Mark rest of the cells initially as 'unvisited'. Then use BFS to find out shortest path of each cell from starting position. If the end cell is visited then we will return "Possible" otherwise "Not Possible".

Algorithm :

1. Take an array of size m*n. Initialize all the cells to 0.
2. For each cell of the rectangle check whether it comes within any circle or not (by calculating the distance of that cell from each circle). If it comes within any circle then change the value of that cell to -1('blocked').

3. Now, apply BFS from the starting cell and if a cell can be reached then change the value of that cell to 1.
4. If the value of the ending cell is 1, then return 'Possible', otherwise return 'Not Possible'.

C++

```
 // C++ program to find out path in
// a rectangle containing circles.
#include <iostream>
#include <math.h>
#include <vector>

using namespace std;

// Function to find out if there is
// any possible path or not.
bool isPossible(int m, int n, int k, int r,
                vector<int> X, vector<int> Y)
{
  // Take an array of m*n size and
  // initialize each element to 0.
  int rect[m][n] = {0};

  // Now using Pythagorean theorem find if a
  // cell touches or within any circle or not.
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
      for (int p = 0; p < k; p++) {
        if (sqrt((pow((X[p] - 1 - i), 2) +
            pow((Y[p] - 1 - j), 2))) <= r)
        {
          rect[i][j] = -1;
        }
      }
    }
  }

  // If the starting cell comes within
  // any circle return false.
  if (rect[0][0] == -1)
    return false;

  // Now use BFS to find if there
  // is any possible path or not.

  // Initialize the queue which holds
  // the discovered cells whose neighbors
```

```
// are not discovered yet.
vector<vector<int>> qu;

rect[0][0] = 1;
qu.push_back({0, 0});

// Discover cells until queue is not empty
while (!qu.empty()) {
  vector<int> arr = qu.front();
  qu.erase(qu.begin());
  int elex = arr[0];
  int eley = arr[1];

  // Discover the eight adjacent nodes.
  // check top-left cell
  if ((elex > 0) && (eley > 0) &&
      (rect[elex - 1][eley - 1] == 0))
  {
    rect[elex - 1][eley - 1] = 1;
    vector<int> v = {elex - 1, eley - 1};
    qu.push_back(v);
  }

  // check top cell
  if ((elex > 0) &&
      (rect[elex - 1][eley] == 0))
  {
    rect[elex - 1][eley] = 1;
    vector<int> v = {elex - 1, eley};
    qu.push_back(v);
  }

  // check top-right cell
  if ((elex > 0) && (eley < n - 1) &&
      (rect[elex - 1][eley + 1] == 0))
  {
    rect[elex - 1][eley + 1] = 1;
    vector<int> v = {elex - 1, eley + 1};
    qu.push_back(v);
  }

  // check left cell
  if ((eley > 0) &&
      (rect[elex][eley - 1] == 0))
  {
    rect[elex][eley - 1] = 1;
    vector<int> v = {elex, eley - 1};
    qu.push_back(v);
```

```
    }

    // check right cell
    if ((eley > n - 1) &&
        (rect[elex][eley + 1] == 0))
    {
      rect[elex][eley + 1] = 1;
      vector<int> v = {elex, eley + 1};
      qu.push_back(v);
    }

    // check bottom-left cell
    if ((elex < m - 1) && (eley > 0) &&
        (rect[elex + 1][eley - 1] == 0))
    {
      rect[elex + 1][eley - 1] = 1;
      vector<int> v = {elex + 1, eley - 1};
      qu.push_back(v);
    }

    // check bottom cell
    if ((elex < m - 1) &&
        (rect[elex + 1][eley] == 0))
    {
      rect[elex + 1][eley] = 1;
      vector<int> v = {elex + 1, eley};
      qu.push_back(v);
    }

    // check bottom-right cell
    if ((elex < m - 1) && (eley < n - 1) &&
        (rect[elex + 1][eley + 1] == 0))
    {
      rect[elex + 1][eley + 1] = 1;
      vector<int> v = {elex + 1, eley + 1};
      qu.push_back(v);
    }
  }

  // Now if the end cell (i.e. bottom right cell)
  // is 1(reachable) then we will send true.
  return (rect[m - 1][n - 1] == 1);
}

// Driver Program
int main() {

  // Test case 1
```

```
  int m1 = 5, n1 = 5, k1 = 2, r1 = 1;
  vector<int> X1 = {1, 3};
  vector<int> Y1 = {3, 3};
  if (isPossible(m1, n1, k1, r1, X1, Y1))
    cout << "Possible" << endl;
  else
    cout << "Not Possible" << endl;

  // Test case 2
  int m2 = 5, n2 = 5, k2 = 2, r2 = 1;
  vector<int> X2 = {1, 1};
  vector<int> Y2 = {2, 3};
  if (isPossible(m2, n2, k2, r2, X2, Y2))
    cout << "Possible" << endl;
  else
    cout << "Not Possible" << endl;

  return 0;
}
```

Output:

```
Possible
Not Possible
```

Time Complexity : It takes O(m*n*k) time to compute whether a cell is within or not in any circle. And it takes O(V+E) time in BFS. Here, number of edges in m*n grid is m*(n-1)+n*(m-1) and vertices m*n. So it takes O(m*n) time in DFS. Hence, the time complexity is O(m*n*k). The complexity can be improved if we iterate through each circles and mark -1 the cells which are coming within it.

Source

https://www.geeksforgeeks.org/path-rectangle-containing-circles/

Chapter 196

Previous greater element

Previous greater element - GeeksforGeeks

Given an array of distinct elements, find previous greater element for every element. If previous greater element does not exist, print -1.

Examples:

```
Input : arr[] = {10, 4, 2, 20, 40, 12, 30}
Output :         -1, 10, 4, -1, -1, 40, 40

Input : arr[] = {10, 20, 30, 40}
Output :        -1, -1, -1, -1

Input : arr[] = {40, 30, 20, 10}
Output :        -1, 40, 30, 20
```

Expected time complexity : O(n)

A **simple solution** is to run two nested loops. The outer loop picks an element one by one. The inner loop, find the previous element that is greater.

C++

```
 // C++ program previous greater element
// A naive solution to print previous greater
// element for every element in an array.
#include <bits/stdc++.h>
using namespace std;

void prevGreater(int arr[], int n)
{
    // Previous greater for first element never
```

```
    // exists, so we print -1.
    cout << "-1, ";

    // Let us process remaining elements.
    for (int i = 1; i < n; i++) {

        // Find first element on left side
        // that is greater than arr[i].
        int j;
        for (j = i-1; j >= 0; j--) {
            if (arr[i] < arr[j]) {
            cout << arr[j] << ", ";
            break;
            }
        }

        // If all elements on left are smaller.
        if (j == -1)
        cout << "-1, ";
    }
}
// Driver code
int main()
{
    int arr[] = { 10, 4, 2, 20, 40, 12, 30 };
    int n = sizeof(arr) / sizeof(arr[0]);
    prevGreater(arr, n);
    return 0;
}
```

Java

```
 // Java program previous greater element
// A naive solution to print
// previous greater element
// for every element in an array.
import java.io.*;
import java.util.*;
import java.lang.*;

class GFG
{
static void prevGreater(int arr[],
                        int n)
{
    // Previous greater for
    // first element never
    // exists, so we print -1.
```

```
    System.out.print("-1, ");

    // Let us process
    // remaining elements.
    for (int i = 1; i < n; i++)
    {

        // Find first element on
        // left side that is
        // greater than arr[i].
        int j;
        for (j = i-1; j >= 0; j--)
        {
            if (arr[i] < arr[j])
            {
            System.out.print(arr[j] + ", ");
            break;
            }
        }

        // If all elements on
        // left are smaller.
        if (j == -1)
        System.out.print("-1, ");
    }
}

// Driver Code
public static void main(String[] args)
{
    int arr[] = {10, 4, 2, 20, 40, 12, 30};
    int n = arr.length;
    prevGreater(arr, n);
}
}
```

Python 3

```
 # Python 3 program previous greater element
# A naive solution to print previous greater
# element for every element in an array.
def prevGreater(arr, n) :

    # Previous greater for first element never
    # exists, so we print -1.
    print("-1",end = ", ")

    # Let us process remaining elements.
```

```
    for i in range(1, n) :
        flag = 0

        # Find first element on left side
        # that is greater than arr[i].
        for j in range(i-1, -1, -1) :
            if arr[i] < arr[j] :
                print(arr[j],end = ", ")
                flag = 1
                break

        # If all elements on left are smaller.
        if j == 0 and flag == 0:
            print("-1",end  = ", ")


# Driver code
if __name__ == "__main__" :
    arr = [10, 4, 2, 20, 40, 12, 30]
    n = len(arr)
    prevGreater(arr, n)

# This code is contributed by ANKITRAI1
```

PHP

```
 <?php
// php program previous greater element
// A naive solution to print previous greater
// element for every element in an array.

function prevGreater(&$arr,$n)
{
    // Previous greater for first element never
    // exists, so we print -1.
    echo( "-1, ");

    // Let us process remaining elements.
    for ($i = 1; $i < $n; $i++)
    {

        // Find first element on left side
        // that is greater than arr[i].
        for ($j = $i-1; $j >= 0; $j--)
    {
            if ($arr[$i] < $arr[$j])
        {
            echo($arr[$j]);
```

```
        echo( ", ");
            break;
            }
        }

        // If all elements on left are smaller.
        if ($j == -1)
        echo("-1, ");
    }
}

// Driver code
$arr = array(10, 4, 2, 20, 40, 12, 30);
$n = sizeof($arr) ;
prevGreater($arr, $n);

//This code is contributed by Shivi_Aggarwal.

?>
```

Output:

```
-1, 10, 4, -1, -1, 40, 40
```

An **efficient solution** is to use stack data structure. If we take a closer look, we can notice that this problem is a variation of stock span problem. We maintain previous greater element in a stack.

C++

```
 // C++ program previous greater element
// An efficient solution to print previous greater
// element for every element in an array.
#include <bits/stdc++.h>
using namespace std;

void prevGreater(int arr[], int n)
{
    // Create a stack and push index of first element
    // to it
    stack<int> s;
    s.push(arr[0]);

    // Previous greater for first element is always -1.
    cout << "-1, ";

    // Traverse remaining elements
```

```
    for (int i = 1; i < n; i++) {

        // Pop elements from stack while stack is not empty
        // and top of stack is smaller than arr[i]. We
        // always have elements in decreasing order in a
        // stack.
        while (s.empty() == false && s.top() < arr[i])
            s.pop();

        // If stack becomes empty, then no element is greater
        // on left side. Else top of stack is previous
        // greater.
        s.empty() ? cout << "-1, " : cout << s.top() << ", ";

        s.push(arr[i]);
    }
}
// Driver code
int main()
{
    int arr[] = { 10, 4, 2, 20, 40, 12, 30 };
    int n = sizeof(arr) / sizeof(arr[0]);
    prevGreater(arr, n);
    return 0;
}
```

Java

```
 // Java program previous greater element
// An efficient solution to
// print previous greater
// element for every element
// in an array.
import java.io.*;
import java.util.*;
import java.lang.*;

class GFG
{
static void prevGreater(int arr[],
                        int n)
{
    // Create a stack and push
    // index of first element
    // to it
    Stack<Integer> s = new Stack<Integer>();
    s.push(arr[0]);
```

```
    // Previous greater for
    // first element is always -1.
    System.out.print("-1, ");

    // Traverse remaining elements
    for (int i = 1; i < n; i++)
    {

        // Pop elements from stack
        // while stack is not empty
        // and top of stack is smaller
        // than arr[i]. We always have
        // elements in decreasing order
        // in a stack.
        while (s.empty() == false &&
            s.peek() < arr[i])
            s.pop();

        // If stack becomes empty, then
        // no element is greater on left
        // side. Else top of stack is
        // previous greater.
        if (s.empty() == true)
            System.out.print("-1, ");
        else
            System.out.print(s.peek() + ", ");

        s.push(arr[i]);
    }
}

// Driver Code
public static void main(String[] args)
{
    int arr[] = { 10, 4, 2, 20, 40, 12, 30 };
    int n = arr.length;
    prevGreater(arr, n);
}
}
```

Output:

```
-1, 10, 4, -1, -1, 40, 40
```

Time Complexity: O(n). It seems more than O(n) at first look. If we take a closer look, we can observe that every element of array is added and removed from stack at most once.

So there are total 2n operations at most. Assuming that a stack operation takes O(1) time, we can say that the time complexity is O(n).

Auxiliary Space: O(n) in worst case when all elements are sorted in decreasing order.

Improved By : Shivi_Aggarwal, ANKITRAI1

Source

https://www.geeksforgeeks.org/previous-greater-element/

Chapter 197

Print all pairs with given sum

Print all pairs with given sum - GeeksforGeeks

Given an array of integers, and a number 'sum', print all pairs in the array whose sum is equal to 'sum'.

```
Examples :
Input  :  arr[] = {1, 5, 7, -1, 5},
          sum = 6
Output : (1, 5) (7, -1) (1, 5)

Input  :  arr[] = {2, 5, 17, -1},
          sum = 7
Output :  (2, 5)
```

A **simple solution** is be traverse each element and check if there's another number in the array which can be added to it to give sum.

C++

```
 // C++ implementation of simple method to
// find print pairs with given sum.
#include <bits/stdc++.h>
using namespace std;

// Returns number of pairs in arr[0..n-1]
// with sum equal to 'sum'
int printPairs(int arr[], int n, int sum)
{
    int count = 0; // Initialize result

    // Consider all possible pairs and check
```

```
    // their sums
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] + arr[j] == sum)
                cout << "(" << arr[i] << ", "
                     << arr[j] << ")" << endl;
}

// Driver function to test the above function
int main()
{
    int arr[] = { 1, 5, 7, -1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int sum = 6;
    printPairs(arr, n, sum);
    return 0;
}
```

Java

```
 // Java implementation of
// simple method to find
// print pairs with given sum.

class GFG
{

// Returns number of pairs
// in arr[0..n-1] with sum
// equal to 'sum'
static void printPairs(int arr[],
                       int n, int sum)
{
    // int count = 0;

    // Consider all possible pairs
    // and check their sums
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] + arr[j] == sum)
                System.out.println( "(" + arr[i] +
                                    ", " + arr[j] +
                                              ")");
}


// Driver Code
public static void main(String []arg)
```

```
{
    int arr[] = {1, 5, 7, -1, 5};
    int n = arr.length;
    int sum = 6;
    printPairs(arr, n, sum);
}
}

// This code is contributed
// by Smitha
```

Python 3

```
 # Python 3 implementation
# of simple method to find
# print pairs with given sum.

# Returns number of pairs
# in arr[0..n-1] with sum
# equal to 'sum'
def printPairs(arr, n, sum):

    # count = 0

    # Consider all possible
    # pairs and check their sums
    for i in range(0, n ):
        for j in range(i + 1, n ):
            if (arr[i] + arr[j] == sum):
                print("(" , arr[i] ,
                      ", ", arr[j],
                      ")", sep = "")


# Driver Code
arr = [1, 5, 7, -1, 5]
n = len(arr)
sum = 6
printPairs(arr, n, sum)

# This code is contributed
# by Smitha
```

C#

```
 // C# implementation of simple
// method to find print pairs
```

```
// with given sum.
using System;

class GFG
{
// Returns number of pairs
// in arr[0..n-1] with sum
// equal to 'sum'
static void printPairs(int []arr,
                       int n, int sum)
{
    // int count = 0;

    // Consider all possible pairs
    // and check their sums
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] + arr[j] == sum)
                Console.Write( "(" + arr[i] +
                               ", " + arr[j] +
                               ")" + "\n");
}


// Driver Code
public static void Main()
{
    int []arr = { 1, 5, 7, -1, 5 };
    int n = arr.Length;
    int sum = 6;
    printPairs(arr, n, sum);
}
}

// This code is contributed
// by Smitha
```

PHP

```
 <?php
// PHP implementation of simple
// method to find print pairs
// with given sum.

// Returns number of pairs in
// arr[0..n-1] with sum equal
// to 'sum'
function printPairs($arr, $n, $sum)
```

```
{
    // Initialize result
    $count = 0;

    // Consider all possible
    // pairs and check their sums
    for ($i = 0; $i < $n; $i++)
        for ( $j = $i + 1; $j < $n; $j++)
            if ($arr[$i] + $arr[$j] == $sum)
                echo "(" , $arr[$i] , ", ",
                           $arr[$j] , ")" ,"\n";
}

// Driver Code
$arr = array (1, 5, 7, -1, 5);
$n = sizeof($arr);
$sum = 6;
printPairs($arr, $n, $sum);

// This code is contributed by m_kit
?>
```

Output :

```
(1, 5)
(1, 5)
(7, -1)
```

Method 2 (Use hashing).
We create an empty hash table. Now we traverse through the array and check for pairs in hash table. If a matching element is found, we print the pair number of times equal to number of occurrences of the matching element.

Note that the worst case of time complexity of this solution is **O(c + n)** where c is count of pairs with given sum.

C++

```
 // C++ implementation of simple method to
// find count of pairs with given sum.
#include <bits/stdc++.h>
using namespace std;

// Returns number of pairs in arr[0..n-1]
// with sum equal to 'sum'
void printPairs(int arr[], int n, int sum)
{
```

```
    // Store counts of all elements in map m
    unordered_map<int, int> m;

    // Traverse through all elements
    for (int i = 0; i < n; i++) {

        // Search if a pair can be formed with
        // arr[i].
        int rem = sum - arr[i];
        if (m.find(rem) != m.end()) {
            int count = m[rem];
            for (int j = 0; j < count; j++)
                cout << "(" << rem << ", "
                     << arr[i] << ")" << endl;
        }
        m[arr[i]]++;
    }
}

// Driver function to test the above function
int main()
{
    int arr[] = { 1, 5, 7, -1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int sum = 6;
    printPairs(arr, n, sum);
    return 0;
}
```

Output :

```
(1, 5)
(7, -1)
(1, 5)
```

Improved By : jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/print-all-pairs-with-given-sum/

Chapter 198

Print all possible sums of consecutive numbers with sum N

Print all possible sums of consecutive numbers with sum N - GeeksforGeeks

Given a number N. The task is to print all possible sums of consecutive numbers that add up to N.

Examples :

```
Input : 100
Output :
9 10 11 12 13 14 15 16
18 19 20 21 22

Input :125
Output :
8 9 10 11 12 13 14 15 16 17
23 24 25 26 27
62 63
```

One important fact is we can not find consecutive numbers above N/2 that adds up to N, because N/2 + (N/2 + 1) would be more than N. So we start from start = 1 till end = N/2 and check for every consecutive sequence whether it adds up to N or not. If it is then we print that sequence and start looking for the next sequence by incrementing start point.

C++

```
// C++ program to print consecutive sequences
```

```
// that add to a given value
#include<bits/stdc++.h>
using namespace std;

void findConsecutive(int N)
{
    // Note that we don't ever have to sum
    // numbers > ceil(N/2)
    int start = 1, end = (N+1)/2;

    // Repeat the loop from bottom to half
    while (start < end)
    {
        // Check if there exist any sequence
        // from bottom to half which adds up to N
        int sum = 0;
        for (int i = start; i <= end; i++)
        {
            sum = sum + i;

            // If sum = N, this means consecutive
            // sequence exists
            if (sum == N)
            {
                // found consecutive numbers! print them
                for (int j = start; j <= i; j++)
                    printf("%d ", j);

                printf("\n");
                break;
            }

            // if sum increases N then it can not exist
            // in the consecutive sequence starting from
            // bottom
            if (sum > N)
                break;
        }
        sum = 0;
        start++;
    }
}

// Driver code
int main(void)
{
    int N = 125;
    findConsecutive(N);
```

```
    return 0;
}
```

Java

```
 // Java program to print
// consecutive sequences
// that add to a given value
import java.io.*;

class GFG
{
static void findConsecutive(int N)
{
    // Note that we don't
    // ever have to sum
    // numbers > ceil(N/2)
    int start = 1;
    int end = (N + 1) / 2;

    // Repeat the loop
    // from bottom to half
    while (start < end)
    {
        // Check if there exist
        // any sequence from
        // bottom to half which
        // adds up to N
        int sum = 0;
        for (int i = start; i <= end; i++)
        {
            sum = sum + i;

            // If sum = N, this means
            // consecutive sequence exists
            if (sum == N)
            {
                // found consecutive
                // numbers! print them
                for (int j = start; j <= i; j++)

                    System.out.print(j + " ");
                    System.out.println();
                break;
            }

            // if sum increases N then
            // it can not exist in the
```

```
            // consecutive sequence
            // starting from bottom
            if (sum > N)
                break;
        }
        sum = 0;
        start++;
    }
}

// Driver code
public static void main (String[] args)
{
    int N = 125;
    findConsecutive(N);
}
}

// This code is contributed by m_kit
```

C#

```
 // C# program to print
// consecutive sequences
// that add to a given value
using System;

class GFG
{
static void findConsecutive(int N)
{
    // Note that we don't
    // ever have to sum
    // numbers > ceil(N/2)
    int start = 1;
    int end = (N + 1) / 2;

    // Repeat the loop
    // from bottom to half
    while (start < end)
    {
        // Check if there exist
        // any sequence from
        // bottom to half which
        // adds up to N
        int sum = 0;
        for (int i = start; i <= end; i++)
        {
```

```
            sum = sum + i;

            // If sum = N, this means
            // consecutive sequence exists
            if (sum == N)
            {
                // found consecutive
                // numbers! print them
                for (int j = start; j <= i; j++)

                    Console.Write(j + " ");
                    Console.WriteLine();
                break;
            }

            // if sum increases N then
            // it can not exist in the
            // consecutive sequence
            // starting from bottom
            if (sum > N)
                break;
        }
        sum = 0;
        start++;
    }
}

// Driver code
static public void Main ()
{
    int N = 125;
    findConsecutive(N);
}
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP program to print consecutive
// sequences that add to a given value

function findConsecutive($N)
{
    // Note that we don't ever have
    // to sum numbers > ceil(N/2)
    $start = 1;
```

```
    $end = ($N + 1) / 2;

    // Repeat the loop from
    // bottom to half
    while ($start < $end)
    {
        // Check if there exist any
        // sequence from bottom to
        // half which adds up to N
        $sum = 0;
        for ($i = $start; $i <= $end; $i++)
        {
            $sum = $sum + $i;

            // If sum = N, this means
            // consecutive sequence exists
            if ($sum == $N)
            {
                // found consecutive
                // numbers! print them
                for ($j = $start; $j <= $i; $j++)
                    echo $j ," ";

                echo "\n";
                break;
            }

            // if sum increases N then it
            // can not exist in the consecutive
            // sequence starting from bottom
            if ($sum > $N)
                break;
        }
        $sum = 0;
        $start++;
    }
}

// Driver code
$N = 125;
findConsecutive($N);

// This code is contributed by ajit
?>
```

Output :

```
8 9 10 11 12 13 14 15 16 17
23 24 25 26 27
62 63
```

Optimized Solution:
In above solution, we keep recalculating sums from start to end, which results in O(N^2) worst case time complexity. This can be avoided by using a precomputed array of sums, or better yet – just keeping track of the sum you have so far and adjusting it depending on how it compares to the desired sum.

Time complexity of below code is O(N).

C++

```
 // Optimized C++ program to find sequences of all consecutive
// numbers with sum equal to N
#include <stdio.h>

void printSums(int N)
{
    int start = 1, end = 1;
    int sum = 1;

    while (start <= N/2)
    {
        if (sum < N)
        {
            end += 1;
            sum += end;
        }
        else if (sum > N)
        {
            sum -= start;
            start += 1;
        }
        else if (sum == N)
        {
            for (int i = start; i <= end; ++i)
                printf("%d ", i);

            printf("\n");
            sum -= start;
            start += 1;
        }
    }
}

// Driver Code
```

```
int main()
{
    printSums(125);
    return 0;
}
```

Java

```
 // Optimized Java program to find
// sequences of all consecutive
// numbers with sum equal to N
import java.io.*;

class GFG {

    static void printSums(int N)
    {
        int start = 1, end = 1;
        int sum = 1;

        while (start <= N/2)
        {
            if (sum < N)
            {
                end += 1;
                sum += end;
            }
            else if (sum > N)
            {
                sum -= start;
                start += 1;
            }
            else if (sum == N)
            {
                for (int i = start;
                         i <= end; ++i)

                    System.out.print(i
                                + " ");

                System.out.println();
                sum -= start;
                start += 1;
            }
        }
    }

    // Driver Code
```

```
    public static void main (String[] args)
    {
            printSums(125);
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // Optimized C# program to find
// sequences of all consecutive
// numbers with sum equal to N
using System;

class GFG {

    static void printSums(int N)
    {
        int start = 1, end = 1;
        int sum = 1;

        while (start <= N/2)
        {
            if (sum < N)
            {
                end += 1;
                sum += end;
            }
            else if (sum > N)
            {
                sum -= start;
                start += 1;
            }
            else if (sum == N)
            {
                for (int i = start;
                        i <= end; ++i)

                    Console.Write(i
                                + " ");

                Console.WriteLine();
                sum -= start;
                start += 1;
            }
        }
    }
```

```
    // Driver Code
    public static void Main ()
    {
            printSums(125);
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// Optimized PHP program to find
// sequences of all consecutive
// numbers with sum equal to N

function printSums($N)
{
    $start = 1; $end = 1;
    $sum = 1;

    while ($start <= $N / 2)
    {
        if ($sum < $N)
        {
            $end += 1;
            $sum += $end;
        }
        else if ($sum > $N)
        {
            $sum -= $start;
            $start += 1;
        }
        else if ($sum == $N)
        {
            for ($i = $start; $i <= $end; ++$i)
                echo $i," ";
                echo "\n";
            $sum -= $start;
            $start += 1;
        }
    }
}

// Driver Code
    printSums(125);
```

```
// This code is contributed by jit_t
?>
```

Output :

```
8 9 10 11 12 13 14 15 16 17
23 24 25 26 27
62 63
```

Reference :
https://www.careercup.com/page?pid=microsoft-interview-questions&n=2

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/print-possible-sums-consecutive-numbers-sum-n/

Chapter 199

Print all triplets in sorted array that form AP

Print all triplets in sorted array that form AP - GeeksforGeeks

Given a sorted array of distinct positive integers, print all triplets that form AP (or Arithmetic Progression)

Examples :

```
Input : arr[] = { 2, 6, 9, 12, 17, 22, 31, 32, 35, 42 };
Output :
6 9 12
2 12 22
12 17 22
2 17 32
12 22 32
9 22 35
2 22 42
22 32 42

Input : arr[] = { 3, 5, 6, 7, 8, 10, 12};
Output :
3 5 7
5 6 7
6 7 8
6 8 10
8 10 12
```

A **simple solution** is to run three nested loops to generate all triplets and for every triplet, check if it forms AP or not. Time complexity of this solution is $O(n^3)$

A **better solution** is to use hashing. We traverse array from left to right. We consider every element as middle and all elements after it as next element. To search the previous element, we use hash table.

C++

```
 // C++ program to print all triplets in given
// array that form Arithmetic Progression
// C++ program to print all triplets in given
// array that form Arithmetic Progression
#include <bits/stdc++.h>
using namespace std;

// Function to print all triplets in
// given sorted array that forms AP
void printAllAPTriplets(int arr[], int n)
{
    unordered_set<int> s;
    for (int i = 0; i < n - 1; i++)
    {
    for (int j = i + 1; j < n; j++)
    {
        // Use hash to find if there is
        // a previous element with difference
        // equal to arr[j] - arr[i]
        int diff = arr[j] - arr[i];
        if (s.find(arr[i] - diff) != s.end())
            cout << arr[i] - diff << " " << arr[i]
                 << " " << arr[j] << endl;
    }
    s.insert(arr[i]);
    }
}

// Driver code
int main()
{
    int arr[] = { 2, 6, 9, 12, 17,
                 22, 31, 32, 35, 42 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printAllAPTriplets(arr, n);
    return 0;
}
```

Java

```
 // Java program to print all
// triplets in given array
```

```
// that form Arithmetic
// Progression
import java.io.*;
import java.util.*;

class GFG
{
    // Function to print
    // all triplets in
    // given sorted array
    // that forms AP
    static void printAllAPTriplets(int []arr,
                                   int n)
    {
        ArrayList<Integer> s =
                 new ArrayList<Integer>();
        for (int i = 0;
                 i < n - 1; i++)
        {
            for (int j = i + 1; j < n; j++)
            {
                // Use hash to find if
                // there is a previous
                // element with difference
                // equal to arr[j] - arr[i]
                int diff = arr[j] - arr[i];
                boolean exists =
                        s.contains(arr[i] - diff);

                if (exists)
                    System.out.println(arr[i] - diff +
                                       " " + arr[i] +
                                       " " + arr[j]);
            }

        s.add(arr[i]);
        }
    }

    // Driver code
    public static void main(String args[])
    {
        int []arr = {2, 6, 9, 12, 17,
                     22, 31, 32, 35, 42};
        int n = arr.length;
        printAllAPTriplets(arr, n);
    }
}
```

```
// This code is contributed by
// Manish Shaw(manishshaw1)
```

Python3

```
 # Python program to prall
# triplets in given array
# that form Arithmetic
# Progression

# Function to print
# all triplets in
# given sorted array
# that forms AP
def printAllAPTriplets(arr, n) :

    s = [];
    for i in range(0, n - 1) :

        for j in range(i + 1, n) :

            # Use hash to find if
            # there is a previous
            # element with difference
            # equal to arr[j] - arr[i]
            diff = arr[j] - arr[i];

            if ((arr[i] - diff) in arr) :
                print ("{} {} {}" .
                        format((arr[i] - diff),
                                arr[i], arr[j]),
                                    end = "\n");

    s.append(arr[i]);

# Driver code
arr = [2, 6, 9, 12, 17,
       22, 31, 32, 35, 42];
n = len(arr);
printAllAPTriplets(arr, n);

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
 // C# program to print all
```

```
// triplets in given array
// that form Arithmetic
// Progression
using System;
using System.Collections.Generic;

class GFG
{
    // Function to print
    // all triplets in
    // given sorted array
    // that forms AP
    static void printAllAPTriplets(int []arr,
                                   int n)
    {
        List<int> s = new List<int>();
        for (int i = 0;
                 i < n - 1; i++)
        {
            for (int j = i + 1; j < n; j++)
            {
                // Use hash to find if
                // there is a previous
                // element with difference
                // equal to arr[j] - arr[i]
                int diff = arr[j] - arr[i];
                bool exists = s.Exists(element =>
                                       element == (arr[i] -
                                                   diff));
                if (exists)
                    Console.WriteLine(arr[i] - diff +
                                      " " + arr[i] +
                                      " " + arr[j]);
            }
        s.Add(arr[i]);
        }
    }

    // Driver code
    static void Main()
    {
        int []arr = new int[]{ 2, 6, 9, 12, 17,
                               22, 31, 32, 35, 42 };
        int n = arr.Length;
        printAllAPTriplets(arr, n);
    }
}
// This code is contributed by
```

```
// Manish Shaw(manishshaw1)
```

PHP

```
 <?php
// PHP program to pr$all
// triplets in given array
// that form Arithmetic
// Progression

// Function to print
// all triplets in
// given sorted array
// that forms AP
function printAllAPTriplets($arr, $n)
{
    $s = array();
    for ($i = 0; $i < $n - 1; $i++)
    {
        for ($j = $i + 1;
             $j < $n; $j++)
        {
            // Use hash to find if
            // there is a previous
            // element with difference
            // equal to arr[j] - arr[i]
            $diff = $arr[$j] - $arr[$i];

            if (in_array($arr[$i] -
                        $diff, $arr))
                echo(($arr[$i] - $diff) .
                        " " . $arr[$i] .
                        " " . $arr[$j] . "\n");
        }
    array_push($s, $arr[$i]);
    }
}

// Driver code
$arr = array(2, 6, 9, 12, 17,
            22, 31, 32, 35, 42);
$n = count($arr);
printAllAPTriplets($arr, $n);

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output :

```
6 9 12
2 12 22
12 17 22
2 17 32
12 22 32
9 22 35
2 22 42
22 32 42
```

Time Complexity : O(n^2)
Auxiliary Space : O(n)

An **efficient solution** is based on the fact that array is sorted. We use the same concept as discussed in GP triplet question. The idea is to start from the second element and fix every element as middle element and search for the other two elements in a triplet (one smaller and one greater).

Below is the implementation of the above idea.

C++

```
 // C++ program to print all triplets in given
// array that form Arithmetic Progression
#include <bits/stdc++.h>
using namespace std;

// Function to print all triplets in
// given sorted array that forms AP
void printAllAPTriplets(int arr[], int n)
{
    for (int i = 1; i < n - 1; i++)
    {

        // Search other two elements of
        // AP with arr[i] as middle.
        for (int j = i - 1, k = i + 1; j >= 0 && k < n;)
        {

            // if a triplet is found
            if (arr[j] + arr[k] == 2 * arr[i])
            {
                cout << arr[j] << " " << arr[i]
                     << " " << arr[k] << endl;

                // Since elements are distinct,
```

```
                // arr[k] and arr[j] cannot form
                // any more triplets with arr[i]
                k++;
                j--;
            }

            // If middle element is more move to
            // higher side, else move lower side.
            else if (arr[j] + arr[k] < 2 * arr[i])
                k++;
            else
                j--;
        }
    }
}

// Driver code
int main()
{
    int arr[] = { 2, 6, 9, 12, 17,
                  22, 31, 32, 35, 42 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printAllAPTriplets(arr, n);
    return 0;
}
```

Java

```
 // Java implementation to print
// all the triplets in given array
// that form Arithmetic Progression

import java.io.*;

class GFG
{

    // Function to print all triplets in
    // given sorted array that forms AP
    static void findAllTriplets(int arr[], int n)
    {

        for (int i = 1; i < n - 1; i++)
        {

            // Search other two elements
            // of AP with arr[i] as middle.
            for (int j = i - 1, k = i + 1; j >= 0 && k < n;)
```

```
            {

                // if a triplet is found
                if (arr[j] + arr[k] == 2 * arr[i])
                {
                    System.out.println(arr[j] +" " +
                                       arr[i]+ " " + arr[k]);

                    // Since elements are distinct,
                    // arr[k] and arr[j] cannot form
                    // any more triplets with arr[i]
                    k++;
                    j--;
                }

                // If middle element is more move to
                // higher side, else move lower side.
                else if (arr[j] + arr[k] < 2 * arr[i])
                    k++;
                else
                    j--;
            }
        }
    }

    // Driver code
    public static void main (String[] args)
    {

        int arr[] = { 2, 6, 9, 12, 17,
                      22, 31, 32, 35, 42 };
        int n = arr.length;

        findAllTriplets(arr, n);
    }
}

// This code is contributed by vt_m.
```

Python 3

```
 # python 3 program to print all triplets in given
# array that form Arithmetic Progression

# Function to print all triplets in
# given sorted array that forms AP
def printAllAPTriplets(arr, n):
```

```
    for i in range(1, n - 1):

        # Search other two elements of
        # AP with arr[i] as middle.
        j = i - 1
        k = i + 1
        while(j >= 0 and k < n ):

            # if a triplet is found
            if (arr[j] + arr[k] == 2 * arr[i]):
                print(arr[j], "", arr[i], "", arr[k])

                # Since elements are distinct,
                # arr[k] and arr[j] cannot form
                # any more triplets with arr[i]
                k += 1
                j -= 1


            # If middle element is more move to
            # higher side, else move lower side.
            elif (arr[j] + arr[k] < 2 * arr[i]):
                k += 1
            else:
                j -= 1

# Driver code
arr = [ 2, 6, 9, 12, 17,
        22, 31, 32, 35, 42 ]
n = len(arr)
printAllAPTriplets(arr, n)

# This article is contributed
# by Smitha Dinesh Semwal
```

C#

```
 // C# implementation to print
// all the triplets in given array
// that form Arithmetic Progression

using System;

class GFG
{

    // Function to print all triplets in
    // given sorted array that forms AP
```

```
    static void findAllTriplets(int []arr, int n)
    {

        for (int i = 1; i < n - 1; i++)
        {

            // Search other two elements
            // of AP with arr[i] as middle.
            for (int j = i - 1, k = i + 1; j >= 0 && k < n;)
            {

                // if a triplet is found
                if (arr[j] + arr[k] == 2 * arr[i])
                {
                    Console.WriteLine(arr[j] +" " +
                                      arr[i]+ " " + arr[k]);

                    // Since elements are distinct,
                    // arr[k] and arr[j] cannot form
                    // any more triplets with arr[i]
                    k++;
                    j--;
                }

                // If middle element is more move to
                // higher side, else move lower side.
                else if (arr[j] + arr[k] < 2 * arr[i])
                    k++;
                else
                    j--;
            }
        }
    }

    // Driver code
    public static void Main ()
    {

        int []arr = { 2, 6, 9, 12, 17,
                      22, 31, 32, 35, 42 };
        int n = arr.Length;

        findAllTriplets(arr, n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP implementation to print
// all the triplets in given array
// that form Arithmetic Progression

// Function to print all triplets in
// given sorted array that forms AP
function findAllTriplets($arr, $n)
{
    for ($i = 1; $i < $n - 1; $i++)
    {

        // Search other two elements
        // of AP with arr[i] as middle.
        for ($j = $i - 1, $k = $i + 1;
                  $j >= 0 && $k < $n
        {

            // if a triplet is found
            if ($arr[$j] + $arr[$k] == 2 *
                           $arr[$i])
            {
                echo $arr[$j] ." " .
                     $arr[$i]. " " .
                     $arr[$k] . "\n";

                // Since elements are distinct,
                // arr[k] and arr[j] cannot form
                // any more triplets with arr[i]
                $k++;
                $j--;
            }

            // If middle element is more move to
            // higher side, else move lower side.
            else if ($arr[$j] + $arr[$k] < 2 *
                                    $arr[$i])
                $k++;
            else
                $j--;
        }
    }
}

// Driver code
$arr = array(2, 6, 9, 12, 17,
```

```
            22, 31, 32, 35, 42);

$n = count($arr);
findAllTriplets($arr, $n);

// This code is contributed by Sam007
?>
```

Output :

```
6 9 12
2 12 22
12 17 22
2 17 32
12 22 32
9 22 35
2 22 42
22 32 42
```

Time Complexity : O(n^2)
Auxiliary Space : O(1)

Improved By : nitin mittal, Sam007, manishshaw1

Source

https://www.geeksforgeeks.org/print-triplets-sorted-array-form-ap/

Chapter 200

Print all triplets with given sum

Print all triplets with given sum - GeeksforGeeks

Given an array of distinct elements. The task is to find triplets in array whose sum is equal to a given number.

Examples :

```
Input : arr[] = {0, -1, 2, -3, 1}
        sum = -2
Output : 0  -3  1
         -1  2 -3

Input : arr[] = {1, -2, 1, 0, 5}
        sum = 0
Output : 1 -2  1
```

Method 1 (Simple : $O(n^3)$)
The naive approach is that run three loops and check one by one that sum of three elements is given sum or not If sum of three elements is given sum, then print elements other wise print not found.

C++

```
 // A simple C++ program to find three elements
// whose sum is equal to given sum
#include <bits/stdc++.h>
using namespace std;

// Prints all triplets in arr[] with given sum
void findTriplets(int arr[], int n, int sum)
{
```

```
    for (int i = 0; i < n - 2; i++) {
        for (int j = i + 1; j < n - 1; j++) {
            for (int k = j + 1; k < n; k++) {
                if (arr[i] + arr[j] + arr[k] == sum) {
                    cout << arr[i] << " "
                         << arr[j] << " "
                         << arr[k] << endl;
                }
            }
        }
    }
}

// Driver code
int main()
{
    int arr[] = { 0, -1, 2, -3, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    findTriplets(arr, n, -2);
    return 0;
}
```

Java

```
 // A simple Java program
// to find three elements
// whose sum is equal to
// given sum
import java.io.*;

class GFG
{

// Prints all triplets in
// arr[] with given sum
static void findTriplets(int arr[],
                         int n, int sum)
{
    for (int i = 0;
             i < n - 2; i++)
    {
        for (int j = i + 1;
                 j < n - 1; j++)
        {
            for (int k = j + 1;
                     k < n; k++)
            {
                if (arr[i] + arr[j] + arr[k] == sum)
```

```
                {
                    System.out.println(arr[i]+ " "+
                                       arr[j] +" "+
                                       arr[k] );
                }
            }
        }
    }
}

// Driver code
public static void main (String[] args)
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = arr.length;
    findTriplets(arr, n, -2);
}
}

// This code is contributed by m_kit
```

Python 3

```
 # A simple Python 3 program
# to find three elements
# whose sum is equal to
# given sum

# Prints all triplets in
# arr[] with given sum
def findTriplets(arr, n, sum):

    for i in range(0 , n - 2):
        for j in range(i + 1 , n - 1):
            for k in range(j + 1, n):
                if (arr[i] + arr[j] +
                    arr[k] == sum):
                    print(arr[i], " ",
                          arr[j] ," ",
                          arr[k] , sep = "")

# Driver code
arr = [ 0, -1, 2, -3, 1 ]
n = len(arr)
findTriplets(arr, n, -2)

# This code is contributed
# by Smitha
```

C#

```
 // A simple C# program
// to find three elements
// whose sum is equal to
// given sum
using System;

class GFG
{

// Prints all triplets in
// arr[] with given sum
static void findTriplets(int []arr,
                         int n, int sum)
{
    for (int i = 0;
             i < n - 2; i++)
    {
        for (int j = i + 1;
                 j < n - 1; j++)
        {
            for (int k = j + 1;
                     k < n; k++)
            {
                if (arr[i] + arr[j] + arr[k] == sum)
                {
                    Console.WriteLine(arr[i]+ " "+
                                      arr[j] +" "+
                                      arr[k] );
                }
            }
        }
    }
}

// Driver code
static public void Main ()
{
    int []arr = {0, -1, 2, -3, 1};
    int n = arr.Length;
    findTriplets(arr, n, -2);
}
}

// This code is contributed by akt_mit
```

PHP

```
 <?php
// A simple PHP program to
// find three elements whose
// sum is equal to given sum

// Prints all triplets in
// arr[] with given sum
function findTriplets($arr, $n, $sum)
{
    for ($i = 0; $i < $n - 2; $i++)
    {
        for ($j = $i + 1; $j < $n - 1; $j++)
        {
            for ($k = $j + 1; $k < $n; $k++)
            {
                if ($arr[$i] + $arr[$j] +
                    $arr[$k] == $sum)
                {
                    echo $arr[$i] , " ",
                         $arr[$j] , " ",
                         $arr[$k] , "\n";
                }
            }
        }
    }
}

// Driver code
$arr = array (0, -1, 2, -3, 1);
$n = sizeof($arr);
findTriplets($arr, $n, -2);

// This code is contributed by aj_36
?>
```

Output :

```
0 -3 1
-1 2 -3
```

Time Complexity : $O(n^3)$
Auxiliary Space : $O(1)$

Method 2 (Hashing : $O(n^2)$)
We iterate through every element. For every element arr[i], we find a pair with sum "-arr[i]". This problem reduces to pairs sum and can be solved in O(n) time using hashing.

```
Run a loop from i=0 to n-2
  Create an empty hash table
  Run inner loop from j=i+1 to n-1
      If -(arr[i] + arr[j]) is present in hash table
         print arr[i], arr[j] and -(arr[i]+arr[j])
      Else
         Insert arr[j] in hash table.
```

C++

```
 // C++ program to find triplets in a given
// array whose sum is equal to given sum.
#include <bits/stdc++.h>
using namespace std;

// function to print triplets with given sum
void findTriplets(int arr[], int n, int sum)
{
    for (int i = 0; i < n - 1; i++) {
        // Find all pairs with sum equals to
        // "sum-arr[i]"
        unordered_set<int> s;
        for (int j = i + 1; j < n; j++) {
            int x = sum - (arr[i] + arr[j]);
            if (s.find(x) != s.end())
                printf("%d %d %d\n", x, arr[i], arr[j]);
            else
                s.insert(arr[j]);
        }
    }
}

// Driver code
int main()
{
    int arr[] = { 0, -1, 2, -3, 1 };
    int sum = -2;
    int n = sizeof(arr) / sizeof(arr[0]);
    findTriplets(arr, n, sum);
    return 0;
}
```

Output:

```
-3 0 1
2 -1 -3
```

Time Complexity : $O(n^2)$
Auxiliary Space : O(n)

Method 3 (Sorting : $O(n^2)$)
The above method requires extra space. We can solve in O(1) extra space. The idea is based on method 2 of this post.

```
1. Sort all element of array
2. Run loop from i=0 to n-2.
     Initialize two index variables l=i+1 and r=n-1
4. while (l < r)
     Check sum of arr[i], arr[l], arr[r] is
     given sum or not if sum is 'sum', then print
     the triplet and do l++ and r--.
5. If sum is less than given sum then l++
6. If sum is greater than given sum then r--
7. If not exist in array then print not found.
```

C++

```
 // C++ program to find triplets in a given
// array whose sum is given sum.
#include <bits/stdc++.h>
using namespace std;

// function to print triplets with given sum
void findTriplets(int arr[], int n, int sum)
{
    // sort array elements
    sort(arr, arr + n);

    for (int i = 0; i < n - 1; i++) {
        // initialize left and right
        int l = i + 1;
        int r = n - 1;
        int x = arr[i];
        while (l < r) {
            if (x + arr[l] + arr[r] == sum) {
                // print elements if it's sum is given sum.
                printf("%d %d %d\n", x, arr[l], arr[r]);
                l++;
                r--;
            }

            // If sum of three elements is less
            // than 'sum' then increment in left
            else if (x + arr[l] + arr[r] < sum)
```

```
                l++;

            // if sum is greater than given sum, then
            // decrement in right side
            else
                r--;
        }
    }
}

// Driver code
int main()
{
    int arr[] = { 0, -1, 2, -3, 1 };
    int sum = -2;
    int n = sizeof(arr) / sizeof(arr[0]);
    findTriplets(arr, n, sum);
    return 0;
}
```

Java

```
 // Java program to find triplets
// in a given array whose sum
// is given sum.
import java.io.*;
import java.util.*;

class GFG
{

// function to print
// triplets with given sum
static void findTriplets(int[] arr,
                         int n, int sum)
{
    // sort array elements
    Arrays.sort(arr);

    for (int i = 0;
             i < n - 1; i++)
    {
        // initialize left and right
        int l = i + 1;
        int r = n - 1;
        int x = arr[i];
        while (l < r)
        {
```

```
            if (x + arr[l] + arr[r] == sum)
            {
                // print elements if it's
                // sum is given sum.
                System.out.println(x + " " + arr[l] +
                                        " " + arr[r]);
                l++;
                r--;
            }

            // If sum of three elements
            // is less than 'sum' then
            // increment in left
            else if (x + arr[l] +
                         arr[r] < sum)
                l++;

            // if sum is greater than
            // given sum, then decrement
            // in right side
            else
                r--;
        }
    }
}

// Driver code
public static void main(String args[])
{
    int[] arr = new int[]{ 0, -1, 2, -3, 1 };
    int sum = -2;
    int n = arr.length;
    findTriplets(arr, n, sum);
}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

C#

```
 // C# program to find triplets
// in a given array whose sum
// is given sum.
using System;

class GFG
{
```

```
// function to print
// triplets with given sum
static void findTriplets(int[] arr,
                         int n, int sum)
{
    // sort array elements
    Array.Sort(arr);

    for (int i = 0; i < n - 1; i++)
    {
        // initialize left and right
        int l = i + 1;
        int r = n - 1;
        int x = arr[i];
        while (l < r)
        {
            if (x + arr[l] + arr[r] == sum)
            {
                // print elements if it's
                // sum is given sum.
                Console.WriteLine(x + " " + arr[l] +
                                  " " + arr[r]);
                l++;
                r--;
            }

            // If sum of three elements
            // is less than 'sum' then
            // increment in left
            else if (x + arr[l] +
                         arr[r] < sum)
                l++;

            // if sum is greater than
            // given sum, then decrement
            // in right side
            else
                r--;
        }
    }
}

// Driver code
static int Main()
{
    int[] arr = new int[]{ 0, -1, 2, -3, 1 };
    int sum = -2;
```

```
    int n = arr.Length;
    findTriplets(arr, n, sum);
    return 0;
}
}

// This code is contributed by rahul
```

PHP

```
 <?php
// PHP program to find triplets
// in a given array whose sum
// is given sum.

// function to print triplets
// with given sum
function findTriplets($arr, $n, $sum)
{
    // sort array elements
    sort($arr);

    for ($i = 0; $i < $n - 1; $i++)
    {
        // initialize left and right
        $l = $i + 1;
        $r = $n - 1;
        $x = $arr[$i];
        while ($l < $r)
        {
            if ($x + $arr[$l] +
                $arr[$r] == $sum)
            {
                // print elements if it's
                // sum is given sum.
                echo $x, " ", $arr[$l],
                         " ", $arr[$r], "\n";
                $l++;
                $r--;
            }

            // If sum of three elements
            // is less than 'sum' then
            // increment in left
            else if ($x + $arr[$l] +
                     $arr[$r] < $sum)
                $l++;
```

```
            // if sum is greater
            // than given sum, then
            // decrement in right side
            else
                $r--;
        }
    }
}

// Driver code
$arr = array(0, -1, 2, -3, 1);
$sum = -2;
$n = sizeof($arr);
findTriplets($arr, $n, $sum);

// This code is contributed by ajit
?>
```

Output:

```
-3 -1 2
-3 0 1
```

Time Complexity : O(n^2)
Auxiliary Space : O(1)

Improved By : jit_t, Smitha Dinesh Semwal, mithunkumarmnnit321, Abby_akku

Source

https://www.geeksforgeeks.org/print-all-triplets-with-given-sum/

Chapter 201

Print n smallest elements from given array in their original order

Print n smallest elements from given array in their original order - GeeksforGeeks

We are given an array of m-elements, we need to find n smallest elements from the array but they must be in the same order as they are in given array.

Examples:

```
Input : arr[] = {4, 2, 6, 1, 5},
        n = 3
Output : 4 2 1
Explanation :
1, 2 and 4 are 3 smallest numbers and
4 2 1 is their order in given array.

Input : arr[] = {4, 12, 16, 21, 25},
        n = 3
Output : 4 12 16
Explanation :
4, 12 and 16 are 3 smallest numbers and
4 12 16 is their order in given array.
```

Make a copy of original array and then sort copy array. After sorting the copy array, save all n smallest numbers. Further for each element in original array, check whether it is in n-smallest number or not if it present in n-smallest array then print it otherwise move forward.

Make copy_arr[]

sort(copy_arr)

For all elements in arr[] -

- Find arr[i] in n-smallest element of copy_arr
- If found then print the element

Below is CPP implementation of above approach :

```
 // CPP for printing smallest n number in order
#include <algorithm>
#include <iostream>
using namespace std;

// Function to print smallest n numbers
void printSmall(int arr[], int asize, int n)
{
    // Make copy of array
    vector<int> copy_arr(arr, arr + asize);

    // Sort copy array
    sort(copy_arr.begin(), copy_arr.begin() + asize);

    // For each arr[i] find whether
    // it is a part of n-smallest
    // with binary search
    for (int i = 0; i < asize; ++i)
        if (binary_search(copy_arr.begin(),
                copy_arr.begin() + n, arr[i]))
            cout << arr[i] << " ";
}

// Driver program
int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int asize = sizeof(arr) / sizeof(arr[0]);
    int n = 5;
    printSmall(arr, asize, n);
    return 0;
}
```

Output :

```
1 3 4 2 0
```

For making a copy of array we need space complexity of O(n) and then for sorting we will need complexity of order O(n log n). Further for each element in arr[] we are performing searching in copy_arr[], which will result O(n) for linear search but we can improve it by applying binary search and hence our overall time complexity will be **O(n log n)**.

Source

https://www.geeksforgeeks.org/find-n-smallest-element-given-array-order-array/

Chapter 202

Print number in ascending order which contains 1, 2 and 3 in their digits.

Print number in ascending order which contains 1, 2 and 3 in their digits. - GeeksforGeeks

Given an array of numbers, the task is to print those numbers in ascending order separated by commas which have 1, 2 and 3 in their digits. If no number containing digits 1, 2, and 3 present then print -1.

Examples:

```
Input : numbers[] = {123, 1232, 456, 234, 32145}
Output : 123, 1232, 32145

Input : numbers[] = {9821, 627183, 12, 1234}
Output : 1234, 627183

Input : numbers[] = {12, 232, 456, 234}
Output : -1
```

Asked in : Goldman Sachs
Approach: First finding all the number in from of array which contains **1, 2 & 3** then sort the number according to 1, 2 and 3 and then print it.

CPP

```
 // CPP program to print all number containing
// 1, 2 and 3 in any order.
#include<bits/stdc++.h>
```

```
using namespace std;


// convert the number to string and find
// if it contains 1, 2 & 3.
bool findContainsOneTwoThree(int number)
{
    string str = to_string(number);
    int countOnes = 0, countTwo = 0, countThree = 0;
    for(int i = 0; i < str.length(); i++) {
        if(str[i] == '1') countOnes++;
        else if(str[i] == '2') countTwo++;
        else if(str[i] == '3') countThree++;
    }
    return (countOnes && countTwo && countThree);
}
// prints all the number containing 1, 2, 3
string printNumbers(int numbers[], int n)
{
    vector<int> oneTwoThree;
    for (int i = 0; i < n; i++)
    {
        // check if the number contains 1,
        // 2 & 3 in any order
        if (findContainsOneTwoThree(numbers[i]))
            oneTwoThree.push_back(numbers[i]);
    }

    // sort all the numbers
    sort(oneTwoThree.begin(), oneTwoThree.end());

    string result = "";
    for(auto number: oneTwoThree)
    {
        int value = number;
        if (result.length() > 0)
            result += ", ";

        result += to_string(value);
    }


    return (result.length() > 0) ? result : "-1";
}

// Driver Code
int main() {
    int numbers[] = { 123, 1232, 456, 234, 32145 };
```

```
    int n = sizeof(numbers)/sizeof(numbers[0]);

    string result = printNumbers(numbers, n);
    cout << result;
    return 0;
}
// This code is contributed
// by Sirjan13
```

Java

```
 // Java program to print all number containing
// 1, 2 and 3 in any order.
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;

class GFG {

    // prints all the number containing 1, 2, 3
    // in any order
    private static String printNumbers(int[] numbers)
    {

        ArrayList<Integer> array = new ArrayList<>();
        for (int number : numbers) {

            // check if the number contains 1,
            // 2 & 3 in any order
            if (findContainsOneTwoThree(number))
                array.add(number);
        }

        // sort all the numbers
        Collections.sort(array);

        StringBuffer strbuf = new StringBuffer();
        Iterator it = array.iterator();
        while (it.hasNext()) {

            int value = (int)it.next();
            if (strbuf.length() > 0)
                strbuf.append(", ");

            strbuf.append(Integer.toString(value));
        }
```

```
        return (strbuf.length() > 0) ?
                    strbuf.toString() : "-1";
    }

    // convert the number to string and find
    // if it contains 1, 2 & 3.
    private static boolean findContainsOneTwoThree(
                                          int number)
    {

        String str = Integer.toString(number);
        return (str.contains("1") && str.contains("2") &&
                                    str.contains("3"));
    }

    public static void main(String[] args)
    {
        int[] numbers = { 123, 1232, 456, 234, 32145 };
        System.out.println(printNumbers(numbers));
    }
}
```

Python

```
 # Python program for printing
# all numbers containing 1,2 and 3

def printNumbers(numbers):

    # convert all numbers
    # to strings
    numbers = map(str, numbers)
    result = []
    for num in numbers:

        # check if each number
        # in the list has 1,2 and 3
        if ('1' in num and
            '2' in num and
            '3' in num):
            result.append(num)

    # if there are no
    # valid numbers
    if not result:
        result = ['-1']
```

```
    return sorted(result);

# Driver Code
numbers = [123, 1232, 456,
           234, 32145]
result = printNumbers(numbers)
print ', '.join(num for num in result)

# This code is contributed
# by IshitaTripathi
```

Output:

```
123, 1232, 32145
```

Time Complexity: Time complexity of the above approach is **O(n)**.

Improved By : IshitaTripathi, sirjan13

Source

https://www.geeksforgeeks.org/print-number-ascending-order-contains-1-2-3-digits/

Chapter 203

Print pair with maximum AND value in an array

Print pair with maximum AND value in an array - GeeksforGeeks

Given an array of n positive elements, find the maximum AND value and the pair of elements generating the maximum AND value from the array.
AND is bitwise & operator.

Examples:

```
Input : arr[] = {4, 8, 12, 16}
Output : Pair = 8, 12
         Maximum AND value = 8

Input : arr[] = {4, 8, 16, 2}
Output : Pair = Not Possible
         Maximum AND value = 0
```

Approach:
Finding Maximum AND value is same as Maximum AND value in an array. Our task is to find the pair of elements resulting in obtained AND value. For finding the elements, simply traverse the whole array and find the AND value of each element with the obtained maximum AND value (result) and if **arr[i] & result == result** , that means arr[i] is the element which will generate maximum AND value. Also, in the case if maximum AND value (result) is zero then we should print "Not possible" in that case.

Below is the implementation of above approach:

C++

```
// CPP Program to find pair with
```

```
// maximum AND value
#include <bits/stdc++.h>
using namespace std;

// Utility function to check number of
// elements having set msb as of pattern
int checkBit(int pattern, int arr[], int n)
{
    int count = 0;
    for (int i = 0; i < n; i++)
        if ((pattern & arr[i]) == pattern)
            count++;
    return count;
}

// Function for finding maximum and
// value pair
int maxAND(int arr[], int n)
{
    int res = 0, count;

    // iterate over total of 30bits
    // from msb to lsb
    for (int bit = 31; bit >= 0; bit--) {

        // find the count of element
        // having set msb
        count = checkBit(res | (1 << bit), arr, n);

        // if count >= 2 set particular
        // bit in result
        if (count >= 2)
            res |= (1 << bit);
    }

    // Find the elements
    if (res == 0)
        cout << "Not Possible\n";

    else {

        // print the pair of elements
        cout << "Pair = ";

        count = 0;

        for (int i = 0; i < n && count < 2; i++) {
```

```
            // inc count value after
            // printing element
            if ((arr[i] & res) == res) {
                count++;
                cout << arr[i] << " ";
            }
        }
    }

    // return the result value
    return res;
}

// Driver function
int main()
{
    int arr[] = { 4, 8, 6, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "\nMaximum AND Value = "
         << maxAND(arr, n);
    return 0;
}
```

Java

```
 // Java Program to find pair
// with maximum AND value
import java.util.*;
import java.io.*;

class GFG
{
// Utility function to check number of
// elements having set msb as of pattern
static int checkBit(int pattern, int arr[], int n)
{
    int count = 0;

    for (int i = 0; i < n; i++)
        if ((pattern & arr[i]) == pattern)
            count++;

    return count;
}

// Function for finding maximum and
// value pair
static int maxAND(int arr[], int n)
```

```
{
    int res = 0, count;

    // iterate over total of 30bits
    // from msb to lsb
    for (int bit = 31; bit >= 0; bit--) {

        // find the count of element
        // having set msb
        count = checkBit(res | (1 << bit), arr, n);

        // if count >= 2 set particular
        // bit in result
        if (count >= 2)
            res |= (1 << bit);
    }

    // Find the elements
    if (res == 0)
        System.out.println("Not Possible");

    else {

        // print the pair of elements
        System.out.print("Pair = ");

        count = 0;

        for (int i = 0; i < n && count < 2; i++) {

            // inc count value after
            // printing element
            if ((arr[i] & res) == res) {
                count++;
                System.out.print(arr[i] + " ");
            }
        }
        System.out.println();
    }

    // return the result value
    return res;
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 4, 8, 6, 2 };
```

```
    int n = arr.length;
    System.out.println("Maximum AND Value = "
                                + maxAND(arr, n));

}
}

// This code is contributed by Sahil_Bansall
```

Python 3

```
 # Python 3 Program to find pair with
# maximum AND value

# Utility function to check number of
# elements having set msb as of pattern
def checkBit(pattern, arr, n):

    count = 0
    for i in range(0, n):
        if ((pattern & arr[i]) == pattern):
            count += 1
    return count

# Function for finding maximum and
# value pair
def maxAND(arr, n):

    res = 0

    # iterate over total of 30bits
    # from msb to lsb
    for bit in range(31 ,-1 ,-1) :

        # find the count of element
        # having set msb
        count = checkBit(res | (1 << bit),
                                    arr, n)

        # if count >= 2 set particular
        # bit in result
        if (count >= 2):
            res |= (1 << bit)

    # Find the elements
    if (res == 0):
        print("Not Possible")
```

```
    else:
        # print the pair of elements
        print("Pair = ", end = "")

        count = 0

        i = 0
        while(i < n and count < 2):

            # inc count value after
            # printing element
            if ((arr[i] & res) == res) :
                count+=1
                print(arr[i] , end = " ")
            i += 1

    # return the result value
    return res

# Driver function
arr = [4, 8, 6, 2 ]
n = len(arr)
print("\nMaximum AND Value = ",
                    maxAND(arr, n))

# This code is contributed by Smitha
```

C#

```
 // C# Program to find pair
// with maximum AND value
using System;

class GFG
    {
    // Utility function to check number of
    // elements having set msb as of pattern
    static int checkBit(int pattern, int []arr, int n)
    {
        int count = 0;

        for (int i = 0; i < n; i++)
            if ((pattern & arr[i]) == pattern)
                count++;

        return count;
    }
```

```
// Function for finding maximum and
// value pair
static int maxAND(int []arr, int n)
{
    int res = 0, count;

    // iterate over total of 30bits
    // from msb to lsb
    for (int bit = 31; bit >= 0; bit--) {

        // find the count of element
        // having set msb
        count = checkBit(res | (1 << bit), arr, n);

        // if count >= 2 set particular
        // bit in result
        if (count >= 2)
            res |= (1 << bit);
    }

    // Find the elements
    if (res == 0)
        Console.Write("Not Possible");

    else {

        // print the pair of elements
        Console.Write("Pair = ");

        count = 0;

        for (int i = 0; i < n && count < 2; i++)
        {

            // inc count value after
            // printing element
            if ((arr[i] & res) == res) {
                count++;
                Console.Write(arr[i] + " ");
            }
        }
        Console.WriteLine();
    }

    // return the result value
    return res;
}
```

```
    // Driver code
    public static void Main()
    {
        int []arr = { 4, 8, 6, 2 };
        int n = arr.Length;
        Console.WriteLine("Maximum AND Value = "
                                    + maxAND(arr, n));

    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// php Program to find pair with
// maximum AND value

// Utility function to check number of
// elements having set msb as of pattern
function checkBit($pattern, $arr, $n)
{
    $count = 0;
    for ($i = 0; $i < $n; $i++)
        if (($pattern & $arr[$i]) == $pattern)
            $count++;
    return $count;
}

// Function for finding maximum
// and  value pair
function maxAND($arr, $n)
{
    $res = 0;

    // iterate over total of 30bits
    // from msb to lsb
    for ($bit = 31; $bit >= 0; $bit--)
    {

        // find the count of element
        // having set msb
        $count = checkBit($res | (1 << $bit),
                                    $arr, $n);

        // if count >= 2 set particular
        // bit in result
```

```
        if ($count >= 2)
            $res |= (1 << $bit);
    }

    // Find the elements
    if ($res == 0)
        echo "Not Possible\n";

    else {

        // print the pair of elements
        echo "Pair = ";

        $count = 0;

        for ($i = 0; $i < $n &&
             $count < 2; $i++)
        {

            // inc count value after
            // printing element
            if (($arr[$i] & $res) == $res)
            {
                $count++;
                echo $arr[$i]. " ";
            }
        }
    }

    // return the result value
    return $res;
}

    // Driver code
    $arr = array( 4, 8, 6, 2 );
    $n = sizeof($arr) / sizeof($arr[0]);
    echo "\nMaximum AND Value = " .maxAND($arr, $n);

//This code is contributed by mits
?>
```

Output:

```
Pair = 4 6
Maximum AND value = 4
```

Improved By : Mithun Kumar, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/print-pair-with-maximum-and-value-in-an-array/

Chapter 204

Print reverse string after removing vowels

Print reverse string after removing vowels - GeeksforGeeks

Given a string **s**, print reverse of string and remove the characters from the reversed string where there are vowels in the original string.

Examples:

```
Input : geeksforgeeks
Output : segrfseg
Explanation :
Reversed string is skeegrofskeeg, removing characters
from indexes 1, 2, 6, 9 & 10 (0 based indexing),
we get segrfseg .

Input :duck
Output :kud
```

A **simple solution** is to first reverse the string, then traverse the reversed string and remove vowels.

An **efficient solution** is to do both tasks in one traversal.
Create an empty string r and traverse the original string s and assign the value to the string r. Check whether, at that index, the original string contains a consonant or not. If yes then print the element at that index from string r.

Basic implementation of the above approach :

```
 // CPP Program for removing characters
// from reversed string where vowels are
// present in original string
```

```
#include <bits/stdc++.h>
using namespace std;

// Function for replacing the string
void replaceOriginal(string s, int n)
{
    // initialize a string of length n
    string r(n, ' ');

    // Traverse through all characters of string
    for (int i = 0; i < n; i++) {

        // assign the value to string r
        // from last index of string s
        r[i] = s[n - 1 - i];

        // if s[i] is a consonant then
        // print r[i]
        if (s[i] != 'a' && s[i] != 'e' && s[i] != 'i'
            && s[i] != 'o' && s[i] != 'u') {
            cout << r[i];
        }
    }
    cout << endl;
}

// Driver function
int main()
{
    string s = "geeksforgeeks";
    int n = s.length();
    replaceOriginal(s, n);

    return 0;
}
```

Output:

```
segrfseg
```

Source

https://www.geeksforgeeks.org/print-reverse-string-removing-vowels/

Chapter 205

Print uncommon elements from two sorted arrays

Print uncommon elements from two sorted arrays - GeeksforGeeks

Given two sorted arrays of distinct elements, we need to print those elements from both arrays that are not common. The output should be printed in sorted order.

Examples :

```
Input : arr1[] = {10, 20, 30}
        arr2[] = {20, 25, 30, 40, 50}
Output : 10 25 40 50
We do not print 20 and 30 as these
elements are present in both arrays.

Input : arr1[] = {10, 20, 30}
        arr2[] = {40, 50}
Output : 10 20 30 40 50
```

The idea is based on merge process of merge sort. We traverse both arrays and skip common elements.

C++

```
 // C++ program to find uncommon elements of
// two sorted arrays
#include <iostream>
using namespace std;
```

```
void printUncommon(int arr1[], int arr2[],
                           int n1, int n2)
{

    int i = 0, j = 0, k = 0;
    while (i < n1 && j < n2) {

        // If not common, print smaller
        if (arr1[i] < arr2[j]) {
            cout << arr1[i] << " ";
            i++;
            k++;
        }
        else if (arr2[j] < arr1[i]) {
            cout << arr2[j] << " ";
            k++;
            j++;
        }

        // Skip common element
        else {
            i++;
            j++;
        }
    }

    // printing remaining elements
    while (i < n1) {
        cout << arr1[i] << " ";
        i++;
        k++;
    }
    while (j < n2) {
        cout << arr2[j] << " ";
        j++;
        k++;
    }
}

// Driver code
int main()
{
    int arr1[] = {10, 20, 30};
    int arr2[] = {20, 25, 30, 40, 50};

    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
```

```
    printUncommon(arr1, arr2, n1, n2);

    return 0;
}
```

Java

```
 // Java program to find uncommon elements
// of two sorted arrays
import java.io.*;

class GFG {

    static void printUncommon(int arr1[],
                      int arr2[], int n1, int n2)
    {

        int i = 0, j = 0, k = 0;
        while (i < n1 && j < n2) {

            // If not common, print smaller
            if (arr1[i] < arr2[j]) {
                System.out.print(arr1[i] + " ");
                i++;
                k++;
            }
            else if (arr2[j] < arr1[i]) {
                System.out.print(arr2[j] + " ");
                k++;
                j++;
            }

            // Skip common element
            else {
                i++;
                j++;
            }
        }

        // printing remaining elements
        while (i < n1) {
            System.out.print(arr1[i] + " ");
            i++;
            k++;
        }
        while (j < n2) {
            System.out.print(arr2[j] + " ");
            j++;
```

```
            k++;
        }
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr1[] = { 10, 20, 30 };
        int arr2[] = { 20, 25, 30, 40, 50 };

        int n1 = arr1.length;
        int n2 = arr2.length;

        printUncommon(arr1, arr2, n1, n2);
    }
}

// This code is contributed by vt_m
```

Python3

```
 # Python 3 program to find uncommon
# elements of two sorted arrays


def printUncommon(arr1, arr2, n1, n2) :

    i = 0
    j = 0
    k = 0
    while (i < n1 and j < n2) :

        # If not common, print smaller
        if (arr1[i] < arr2[j]) :
            print( arr1[i] , end= " ")
            i = i + 1
            k = k + 1

        elif (arr2[j] < arr1[i]) :
            print( arr2[j] , end= " ")
            k = k + 1
            j = j + 1

        # Skip common element
        else :
            i = i + 1
            j = j + 1
```

```
    # printing remaining elements
    while (i < n1) :
        print( arr1[i] , end= " ")
        i = i + 1
        k = k + 1

    while (j < n2) :
        print( arr2[j] , end= " ")
        j = j + 1
        k = k + 1


# Driver code
arr1 = [10, 20, 30]
arr2 = [20, 25, 30, 40, 50]

n1 = len(arr1)
n2 = len(arr2)

printUncommon(arr1, arr2, n1, n2)



# This code is contributed
# by Nikita Tiwari.
```

C#

```
 // C# program to find uncommon elements
// of two sorted arrays
using System;
class GFG {

static void printUncommon(int []arr1,
                          int []arr2,
                          int n1,
                          int n2)
    {

        int i = 0, j = 0, k = 0;
        while (i < n1 && j < n2)
        {

            // If not common, print smaller
            if (arr1[i] < arr2[j])
            {
                Console.Write(arr1[i] + " ");
                i++;
```

```
                k++;
            }
            else if (arr2[j] < arr1[i])
            {
                Console.Write(arr2[j] + " ");
                k++;
                j++;
            }

            // Skip common element
            else
            {
                i++;
                j++;
            }
        }

        // printing remaining elements
        while (i < n1)
        {
            Console.Write(arr1[i] + " ");
            i++;
            k++;
        }
        while (j < n2)
        {
            Console.Write(arr2[j] + " ");
            j++;
            k++;
        }
    }

    // Driver Code
    public static void Main()
    {
        int []arr1 = {10, 20, 30};
        int []arr2 = {20, 25, 30, 40, 50};

        int n1 = arr1.Length;
        int n2 = arr2.Length;

        printUncommon(arr1, arr2, n1, n2);
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to find uncommon
// elements of two sorted arrays

function printUncommon($arr1, $arr2,
                       $n1, $n2)
{
    $i = 0;
    $j = 0;
    $k = 0;
    while ($i < $n1 && $j < $n2)
    {

        // If not common, prsmaller
        if ($arr1[$i] < $arr2[$j])
        {
            echo $arr1[$i] . " ";
            $i++;
            $k++;
        }
        else if ($arr2[$j] < $arr1[$i])
        {
            echo $arr2[$j] . " ";
            $k++;
            $j++;
        }

        // Skip common element
        else
        {
            $i++;
            $j++;
        }
    }

    // printing remaining elements
    while ($i < $n1)
    {
        echo $arr1[$i] . " ";
        $i++;
        $k++;
    }
    while ($j < $n2)
    {
        echo $arr2[$j] . " ";
        $j++;
        $k++;
    }
```

```
}

// Driver code
$arr1 = array(10, 20, 30);
$arr2 = array(20, 25, 30, 40, 50);

$n1 = sizeof($arr1) ;
$n2 = sizeof($arr2) ;

printUncommon($arr1, $arr2, $n1, $n2);

// This code is contributed by Anuj_67
?>
```

Output :

```
10 25 40 50
```

Improved By : Sam007, vt_m

Source

https://www.geeksforgeeks.org/print-uncommon-elements-two-sorted-arrays/

Chapter 206

Probability of a key K present in array

Probability of a key K present in array - GeeksforGeeks

Given an array A[] and size of array is N and one another key K. The task is to find the probability that the Key K present in array.

Examples:

```
Input : N = 6
        A[] = { 4, 7, 2, 0, 8, 7, 5 }
        K = 3
Output :0
Since value of k = 3  is not present in array,
hence the probability of 0.

Input :N = 10
       A[] = { 2, 3, 5, 1, 9, 8, 0, 7, 6, 5 }
       K = 5
Output :0.2
```

The probability of can be found out using the below formula:

```
Probability = total number of K present /
                          size of array.
```

First, count the number of K's and then the probability will be the number of K's divided by N i.e. count / N.

Below is the implementation of the above approach:

C++

```
 // C++ code to find the probability of
// search key K present in array
#include <bits/stdc++.h>
using namespace std;

// Function to find the probability
float kPresentProbability(int a[], int n, int k)
{
    float count = 0;

    for (int i = 0; i < n; i++)
        if (a[i] == k)
            count++;

    // find probability
    return count / n;
}

// Driver Code
int main()
{

    int A[] = { 4, 7, 2, 0, 8, 7, 5 };
    int K = 3;
    int N = sizeof(A) / sizeof(A[0]);
    cout << kPresentProbability(A, N, K);
    return 0;
}
```

Python3

```
 # Python3 code to find the
# probability of search key
# K present in 1D-array (list).

# Function to find the probability
def kPresentProbability(a, n, k) :

    count = a.count(k)

    # find probability upto
    # 2 decimal places
    return round(count / n , 2)

# Driver Code
if __name__ == "__main__" :

    A = [ 4, 7, 2, 0, 8, 7, 5 ]
```

```
    K = 2
    N = len(A)

    print(kPresentProbability( A, N, K))

# This code is contributed
# by AnkitRai1
```

Java

```
 // Java code to find the probability
// of search key K present in array
class GFG
{

// Function to find the probability
static float kPresentProbability(int a[],
                                 int n,
                                 int k)
{
    float count = 0;

    for (int i = 0; i < n; i++)
        if (a[i] == k)
            count++;

    // find probability
    return count/ n;
}

// Driver Code
public static void main(String[] args)
{
    int A[] = { 4, 7, 2, 0, 8, 7, 5 };
    int K = 2;
    int N = A.length;
    double n = kPresentProbability(A, N, K);
    double p = (double)Math.round(n * 100) / 100;
    System.out.println(p);
}
}

// This code is contributed
// by ChitraNayal
```

C#

```
 // C# code to find the probability
```

```
// of search key K present in array
using System;

class GFG
{

// Function to find the probability
static float kPresentProbability(int[] a,
                                 int n,
                                 int k)
{
    float count = 0;

    for (int i = 0; i < n; i++)
        if (a[i] == k)
            count++;

    // find probability
    return count/ n;
}

// Driver Code
public static void Main()
{
    int[] A = { 4, 7, 2, 0, 8, 7, 5 };
    int K = 2;
    int N = A.Length;
    double n = kPresentProbability(A, N, K);
    double p = (double)Math.Round(n * 100) / 100;
    Console.Write(p);
}
}

// This code is contributed
// by ChitraNayal
```

PHP

```
 <?php
// PHP code to find the probability
// of search key K present in array

// Function to find the probability
function kPresentProbability(&$a, $n, $k)
{
    $count = 0;

    for ($i = 0; $i < $n; $i++)
```

```
        if ($a[$i] == $k)
            $count++;

    // find probability
    return $count / $n;
}

// Driver Code
$A = array( 4, 7, 2, 0, 8, 7, 5 );
$K = 2;
$N = sizeof($A);
echo round(kPresentProbability($A, $N, $K), 2);

// This code is contributed
// by ChitraNayal
?>
```

Output:

```
0.14
```

Time Complexity: O(N)

Improved By : ANKITRAI1, ChitraNayal

Source

https://www.geeksforgeeks.org/probability-of-a-key-k-present-in-array/

Chapter 207

Probability of a random pair being the maximum weighted pair

Probability of a random pair being the maximum weighted pair - GeeksforGeeks

Given two arrays A and B, a random pair is picked having an element from array A and another from array B. Output the probability of the pair being maximum weighted.

Examples:

```
Input : A[] = 1 2 3
        B[] = 1 3 3
Output : 0.222
Explanation : Possible pairs are : {1, 1},
{1, 3}, {1, 3}, {2, 1}, {2, 3}, {2, 3},
{3, 1}, {3, 3}, {3, 3} i.e. 9.
The pair with maximum weight is {3, 3} with
frequency 2. So, the probability of random
pair being maximum is 2/9 = 0.2222.
```

Brute Force Method : Generate all possible pairs in N^2 time complexity and count maximum weighted pairs.

Better Method : Sort both the arrays and count the last (max) elements from A and B. No. of maximum weighted pairs will be product of both counts. The probability will be (product of counts) / sizeof(A) * sizeof(B)

Best Method Best approach will be to traverse both the arrays and count the maximum element. No. of maximum weighted pairs will be product of both counts. The probability will be (product of counts) / sizeof(A) * sizeof(B)

Below is the implementation:

C++

```
 #include <bits/stdc++.h>
using namespace std;

// Function to return probability
double probability(int a[], int b[], int size1,
                                      int size2)
{
    // Count occurrences of maximum element
    // in A[]
    int max1 = INT_MIN,  count1 = 0;
    for (int i = 0; i < size1; i++) {
        if (a[i] > max1) {
            max1 = a[i];
            count1 = 1;
        }
        else if (a[i] == max1) {
            count1++;
        }
    }

    // Count occurrences of maximum element
    // in B[]
    int max2 = INT_MIN, count2 = 0;
    for (int i = 0; i < size2; i++) {
        if (b[i] > max2) {
            max2 = b[i];
            count2 = 1;
        }
        else if (b[i] == max2) {
            count2++;
        }
    }

    // Returning probability
    return (double)(count1 * count2) /
                  (size1 * size2);
}

// Driver code
int main()
{
    int a[] = { 1, 2, 3 };
    int b[] = { 1, 3, 3 };
```

```
    int size1 = sizeof(a) / sizeof(a[0]);
    int size2 = sizeof(b) / sizeof(b[0]);

    cout << probability(a, b, size1, size2);
    return 0;
}
```

Java

```
 // Java program to find Probability
// of a random pair being the maximum
// weighted pair
import java.io.*;

class GFG {

    // Function to return probability
    static double probability(int a[], int b[],
                              int size1,int size2)
    {
        // Count occurrences of maximum
        // element in A[]
        int max1 = Integer.MIN_VALUE,  count1 = 0;
        for (int i = 0; i < size1; i++) {
            if (a[i] > max1) {
                max1 = a[i];
                count1 = 1;
            }
            else if (a[i] == max1) {
                count1++;
            }
        }

        // Count occurrences of maximum
        // element in B[]
        int max2 = Integer.MIN_VALUE, count2 = 0;
        for (int i = 0; i < size2; i++) {
            if (b[i] > max2) {
                max2 = b[i];
                count2 = 1;
            }
            else if (b[i] == max2) {
                count2++;
            }
        }

        // Returning probability
        return (double)(count1 * count2) / (size1 * size2);
```

```
    }

    // Driver code
    public static void main(String args[])
    {
        int a[] = { 1, 2, 3 };
        int b[] = { 1, 3, 3 };

        int size1 = a.length;
        int size2 = b.length;

        System.out.println(probability(a, b,
                            size1, size2));
    }
}

/*This code is contributed by Nikita Tiwari.*/
```

C#

```
 // C# program to find Probability of a random
// pair being the maximum weighted pair
using System;

class GFG {

    // Function to return probability
    static float probability(int []a, int []b,
                          int size1,int size2)
    {

        // Count occurrences of maximum
        // element in A[]
        int max1 = int.MinValue, count1 = 0;

        for (int i = 0; i < size1; i++) {
            if (a[i] > max1) {
                max1 = a[i];
                count1 = 1;
            }
            else if (a[i] == max1) {
                count1++;
            }
        }

        // Count occurrences of maximum
        // element in B[]
        int max2 = int.MinValue, count2 = 0;
```

```
        for (int i = 0; i < size2; i++) {
            if (b[i] > max2) {
                max2 = b[i];
                count2 = 1;
            }
            else if (b[i] == max2) {
                count2++;
            }
        }

        // Returning probability
        return (float)(count1 * count2) /
                            (size1 * size2);
    }

    // Driver code
    public static void Main()
    {
        int []a = { 1, 2, 3 };
        int []b = { 1, 3, 3 };

        int size1 = a.Length;
        int size2 = b.Length;

        Console.WriteLine(probability(a, b,
                            size1, size2));
    }
}

/* This code is contributed by vt_m.*/
```

PHP

```
 <?php
// PHP program for Probability of
// a random pair being the maximum
// weighted pair

// Function to return probability
function probability($a, $b,
             $size1, $size2)
{

    // Count occurrences of maximum
    // element in A[]
    $max1 = PHP_INT_MIN; $count1 = 0;
    for ($i = 0; $i < $size1; $i++)
```

```
    {
        if ($a[$i] > $max1)
        {
            $max1 = $a[$i];
            $count1 = 1;
        }
        else if ($a[$i] == $max1)
        {
            $count1++;
        }
    }

    // Count occurrences of maximum
    // element in B[]
    $max2 = PHP_INT_MIN; $count2 = 0;
    for ($i = 0; $i < $size2; $i++)
    {
        if ($b[$i] > $max2)
        {
            $max2 = $b[$i];
            $count2 = 1;
        }
        else if ($b[$i] == $max2)
        {
            $count2++;
        }
    }

    // Returning probability
    return (double)($count1 * $count2) /
                    ($size1 * $size2);
}

    // Driver code
    $a = array(1, 2, 3);
    $b = array(1, 3, 3);
    $size1 = sizeof($a);
    $size2 = sizeof($b);
    echo probability($a, $b,
            $size1, $size2);

// This code is contributed by ajit
?>
```

Output:

```
0.222222
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/probability-random-pair-maximum-weighted-pair/

Chapter 208

Probability of choosing a random pair with maximum sum in an array

Probability of choosing a random pair with maximum sum in an array - GeeksforGeeks

Given an array of N integers, You have to find the probability of choosing a random pair(i, j), i < j such that A[i] + A[j] is maximum.

Examples :

```
Input : A[] = {3, 3, 3, 3}
Output : 1.0000
Explanation :
Here, maximum sum we can get by selecting
any pair is 6.
Total number of pairs possible = 6.
Pairs with maximum sum = 6.
Probability = 6/6 = 1.0000

Input : A[] = {1, 2, 2, 3}
Output : 0.3333
Explanation :
Here, maximum sum we can get by selecting
a pair is 5.
Total number of pairs possible = 6.
Pairs with maximum sum = {2, 3} and {2, 3} = 2.
Probability = 2/6 = 0.3333

Probability(event) = Number of favorable outcomes /
                     Total number of outcomes
```

Naive approach : We can solve this problem using brute force solution overall pair (i, j), i < j to get the maximum value possible and then again do a brute force to calculate the number of times the maximum is attained.

Efficient Approach : Observe that we get maximum pair sum only when the pairs consists of first and second maximum elements of the array. So, the problem is to calculate the number of occurrences of those elements and calculate the favorable outcomes using a formula.

```
Favorable outcomes = f2 (frequency of second maximum
element(f2), if maximum element occurs only once).
      or
Favorable outcomes = f1 * (f1 - 1) / 2,
(when frequency of maximum element(f1)
is greater than 1).
```

C++

```
 // CPP program of choosing a random pair
// such that A[i]+A[j] is maximum.
#include <bits/stdc++.h>
using namespace std;

// Function to get max first and second
int countMaxSumPairs(int a[], int n)
{
    int first = INT_MIN, second = INT_MIN;
    for (int i = 0; i < n; i++) {

        /* If current element is smaller than
          first,  then update both first and
          second */
        if (a[i] > first) {
            second = first;
            first = a[i];
        }

        /* If arr[i] is in between first and
        second then update second */
        else if (a[i] > second && a[i] != first)
            second = a[i];
    }

    int cnt1 = 0, cnt2 = 0;
    for (int i = 0; i < n; i++) {
        if (a[i] == first)
            cnt1++; // frequency of first maximum
        if (a[i] == second)
```

```
            cnt2++; // frequency of second maximum
    }
    if (cnt1 == 1)
        return cnt2;

    if (cnt1 > 1)
        return cnt1 * (cnt1 - 1) / 2;
}

// Returns probability of choosing a pair with
// maximum sum.
float findMaxSumProbability(int a[], int n)
{
    int total = n * (n - 1) / 2;
    int max_sum_pairs = countMaxSumPairs(a, n);
    return (float)max_sum_pairs/(float)total;
}

// Driver Code
int main()
{
    int a[] = { 1, 2, 2, 3 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << findMaxSumProbability(a, n);
    return 0;
}
```

C#

```
 // C# program of choosing a random pair
// such that A[i]+A[j] is maximum.
using System;

public class GFG{


    // Function to get max first and second
    static int countMaxSumPairs(int []a, int n)
    {
        int first = int.MinValue, second = int.MinValue;
        for (int i = 0; i < n; i++) {

            /* If current element is smaller than
            first, then update both first and
            second */
            if (a[i] > first)
            {
                second = first;
```

```
                first = a[i];
            }

            /* If arr[i] is in between first and
            second then update second */
            else if (a[i] > second && a[i] != first)
                second = a[i];
        }

        int cnt1 = 0, cnt2 = 0;
        for (int i = 0; i < n; i++) {
            if (a[i] == first)

                // frequency of first maximum
                cnt1++;
            if (a[i] == second)

                // frequency of second maximum
                cnt2++;
        }
        if (cnt1 == 1)
            return cnt2;

        if (cnt1 > 1)
            return cnt1 * (cnt1 - 1) / 2;
            return 0;
    }

    // Returns probability of choosing a pair with
    // maximum sum.
    static float findMaxSumProbability(int []a, int n)
    {
        int total = n * (n - 1) / 2;
        int max_sum_pairs = countMaxSumPairs(a, n);
        return (float)max_sum_pairs/(float)total;
    }

    // Driver Code
    static public void Main ()
    {
        int []a = { 1, 2, 2, 3 };
        int n = a.Length;;
        Console.WriteLine(findMaxSumProbability(a, n));

    }
}
// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program of choosing a random pair
// such that A[i]+A[j] is maximum.

// Function to get max first and second
function countMaxSumPairs($a, $n)
{
    $first = PHP_INT_MIN;
    $second = PHP_INT_MIN;
    for ($i = 0; $i < $n; $i++)
    {

        // If current element is
        // smaller than first,
        // then update both first
        // and second
        if ($a[$i] > $first)
        {
            $second = $first;
            $first = $a[$i];
        }

        // If arr[i] is in between
        // first and second then
        // update second
        else if ($a[$i] > $second &&
                 $a[$i] != $first)
            $second = $a[$i];
    }

    $cnt1 = 0;
    $cnt2 = 0;
    for ($i = 0; $i < $n; $i++)
    {
        if ($a[$i] == $first)

            // frequency of first maximum
            $cnt1++;
        if ($a[$i] == $second)

            // frequency of second maximum
            $cnt2++;
    }
    if ($cnt1 == 1)
        return $cnt2;
```

```
    if ($cnt1 > 1)
        return $cnt1 * ($cnt1 - 1) / 2;
}

// Returns probability of
// choosing a pair with
// maximum sum.
function findMaxSumProbability($a, $n)
{
    $total = $n * ($n - 1) / 2;
    $max_sum_pairs = countMaxSumPairs($a, $n);
    return (float)$max_sum_pairs / (float) $total;
}

    // Driver Code
    $a= array (1, 2, 2, 3 );
    $n = sizeof($a);
    echo findMaxSumProbability($a, $n);

// This code is contributed by ajit
?>
```

Output :

```
0.333333
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/probability-choosing-random-pair-maximum-sum-array/

Chapter 209

Product of maximum in first array and minimum in second

Product of maximum in first array and minimum in second - GeeksforGeeks

Given two arrays, the task is to calculate the product of max element of first array and min element of second array

References : Asked in Adobe (Source : Careercup)

Examples :

```
Input : arr1[] = {5, 7, 9, 3, 6, 2},
        arr2[] = {1, 2, 6, -1, 0, 9}
Output : max element in first array
is 9 and min element in second array
is -1. The product of these two is -9.

Input : arr1[] = {1, 4, 2, 3, 10, 2},
        arr2[] = {4, 2, 6, 5, 2, 9}
Output : max element in first array
is 10 and min element in second array
is 2. The product of these two is 20.
```

Method 1: Naive approach We first sort both arrays. Then we easily find max in first array and min in second array. Finally we return product of min and max.

C++

```
 // C++ program to calculate the
// product of max element of
// first array and min element
// of second array
```

```
#include <bits/stdc++.h>
using namespace std;

// Function to calculate
// the product
int minMaxProduct(int arr1[],
                  int arr2[],
                  int n1,
                  int n2)
{
    // Sort the arrays to find
    // the maximum and minimum
    // elements in given arrays
    sort(arr1, arr1 + n1);
    sort(arr2, arr2 + n2);

    // Return product of
    // maximum and minimum.
    return arr1[n1 - 1] * arr2[0];
}

// Driven code
int main()
{
    int arr1[] = { 10, 2, 3, 6, 4, 1 };
    int arr2[] = { 5, 1, 4, 2, 6, 9 };
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    int n2 = sizeof(arr1) / sizeof(arr1[0]);
    cout << minMaxProductt(arr1, arr2, n1, n2);
    return 0;
}
```

Java

```
 // Java program to find the
// to calculate the product
// of max element of first
// array and min element of
// second array
import java.util.*;
import java.lang.*;

class GfG
{

    // Function to calculate
    // the product
    public static int minMaxProduct(int arr1[],
```

```
                                    int arr2[],
                                    int n1,
                                    int n2)
    {

        // Sort the arrays to find the
        // maximum and minimum elements
        // in given arrays
        Arrays.sort(arr1);
        Arrays.sort(arr2);

        // Return product of maximum
        // and minimum.
        return arr1[n1 - 1] * arr2[0];
    }

    // Driver Code
    public static void main(String argc[])
    {
        int [] arr1= new int []{ 10, 2, 3,
                                 6, 4, 1 };
        int [] arr2 = new int []{ 5, 1, 4,
                                 2, 6, 9 };
        int n1 = 6;
        int n2 = 6;
        System.out.println(minMaxProduct(arr1,
                                         arr2,
                                         n1, n2));
    }
}

/*This code is contributed by Sagar Shukla.*/
```

Python

```
 # A Python program to find the to
# calculate the product of max
# element of first array and min
# element of second array

# Function to calculate the product
def minmaxProduct(arr1, arr2, n1, n2):

    # Sort the arrays to find the
    # maximum and minimum elements
    # in given arrays
    arr1.sort()
    arr2.sort()
```

```
    # Return product of maximum
    # and minimum.
    return arr1[n1 - 1] * arr2[0]

# Driver Program
arr1 = [10, 2, 3, 6, 4, 1]
arr2 = [5, 1, 4, 2, 6, 9]
n1 = len(arr1)
n2 = len(arr2)
print(minmaxProduct(arr1, arr2, n1, n2))

# This code is contributed by Shrikant13.
```

C#

```
 // C# program to find the to
// calculate the product of
// max element of first array
// and min element of second array
using System;

class GfG
{

    // Function to calculate the product
    public static int minMaxProduct(int []arr1,
                                    int []arr2,
                                    int n1,
                                    int n2)
    {

        // Sort the arrays to find the
        // maximum and minimum elements
        // in given arrays
        Array.Sort(arr1);
        Array.Sort(arr2);

        // Return product of maximum
        // and minimum.
        return arr1[n1 - 1] * arr2[0];
    }

    // Driver Code
    public static void Main()
    {
        int [] arr1= new int []{ 10, 2, 3,
                                 6, 4, 1 };
```

```
        int [] arr2 = new int []{ 5, 1, 4,
                                  2, 6, 9 };
        int n1 = 6;
        int n2 = 6;
        Console.WriteLine(minMaxProduct(arr1, arr2,
                                        n1, n2));
    }
}

/*This code is contributed by vt_m.*/
```

PHP

```
 <?php
// PHP program to find the to
// calculate the product of max
// element of first array and
// min element of second array


// Function to calculate the product
function minMaxProduct( $arr1, $arr2,
                        $n1, $n2)
{
    // Sort the arrays to find
    // the maximum and minimum
    // elements in given arrays
    sort($arr1);
    sort($arr2);

    // Return product of
    // maximum and minimum.
    return $arr1[$n1 - 1] * $arr2[0];
}

// Driver code
$arr1 = array( 10, 2, 3, 6, 4, 1 );
$arr2 = array( 5, 1, 4, 2, 6, 9 );
$n1 = count($arr1);
$n2 = count($arr2);
echo minMaxProduct($arr1, $arr2,
                   $n1, $n2);

// This code is contributed by anuj_67.
?>
```

Output :

10

Time Complexity : O(n log n)
Space Complexity : O(1)

Method 2 : Efficient approach In this approach, we simply traverse the whole arrays and find max in first array and min in second array and can easily get product of min and max.

C++

```
 // C++ program to find the to
// calculate the product of
// max element of first array
// and min element of second array
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the product
int minMaxProduct(int arr1[], int arr2[],
                  int n1, int n2)
{
    // Initialize max of first array
    int max = arr1[0];

    // initialize min of second array
    int min = arr2[0];

    int i;
    for (i = 1; i < n1 && i < n2; ++i)
    {

        // To find the maximum
        // element in first array
        if (arr1[i] > max)
            max = arr1[i];

        // To find the minimum
        // element in second array
        if (arr2[i] < min)
            min = arr2[i];
    }

    // Process remaining elements
    while (i < n1)
    {
        if (arr1[i] > max)
        max = arr1[i];
```

```
        i++;
    }
    while (i < n2)
    {
        if (arr2[i] < min)
        min = arr2[i];
        i++;
    }

    return max * min;
}

// Driven code
int main()
{
    int arr1[] = { 10, 2, 3, 6, 4, 1 };
    int arr2[] = { 5, 1, 4, 2, 6, 9 };
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    int n2 = sizeof(arr1) / sizeof(arr1[0]);
    cout << minMaxProduct(arr1, arr2, n1, n2)
         << endl;
    return 0;
}
```

Java

```
 // Java program to calculate the
// product of max element of first
// array and min element of second array
import java.util.*;
import java.lang.*;

class GfG
{

    // Function to calculate the product
    public static int minMaxProduct(int arr1[],
                                    int arr2[],
                                    int n1,
                                    int n2)
       {

        // Initialize max of
        // first array
        int max = arr1[0];

        // initialize min of
        // second array
```

```
        int min = arr2[0];

        int i;
        for (i = 1; i < n1 && i < n2; ++i)
        {

        // To find the maximum
        // element in first array
        if (arr1[i] > max)
            max = arr1[i];

        // To find the minimum element
        // in second array
        if (arr2[i] < min)
            min = arr2[i];
        }

        // Process remaining elements
        while (i < n1)
        {
            if (arr1[i] > max)
            max = arr1[i];
            i++;
        }
        while (i < n2)
        {
            if (arr2[i] < min)
            min = arr2[i];
            i++;
        }

        return max * min;
    }

    // Driver Code
    public static void main(String argc[])
    {
        int [] arr1= new int []{ 10, 2, 3,
                                 6, 4, 1 };
        int [] arr2 = new int []{ 5, 1, 4,
                                  2, 6, 9 };
        int n1 = 6;
        int n2 = 6;
        System.out.println(minMaxProduct(arr1, arr2,
                                         n1, n2));
    }
}
```

```
// This code is contributed by Sagar Shukla
```

C#

```
 // C# program to find the to
// calculate the product of
// max element of first array
// and min element of second array
using System;

class GfG
{

    // Function to calculate
    // the product
    public static int minMaxProduct(int []arr1,
                                    int []arr2,
                                    int n1,
                                    int n2)
    {

        // Initialize max of
        // first array
        int max = arr1[0];

        // initialize min of
        // second array
        int min = arr2[0];

        int i;
        for (i = 1; i < n1 && i < n2; ++i)
        {

            // To find the maximum element
            // in first array
            if (arr1[i] > max)
                max = arr1[i];

            // To find the minimum element
            // in second array
            if (arr2[i] < min)
                min = arr2[i];
        }

        // Process remaining elements
        while (i < n1)
        {
            if (arr1[i] > max)
```

```
            max = arr1[i];
            i++;
        }
        while (i < n2)
        {
            if (arr2[i] < min)
            min = arr2[i];
            i++;
        }

        return max * min;
    }

    // Driver Code
    public static void Main()
    {
        int [] arr1= new int []{ 10, 2, 3,
                                 6, 4, 1 };
        int [] arr2 = new int []{ 5, 1, 4,
                                  2, 6, 9 };
        int n1 = 6;
        int n2 = 6;
        Console.WriteLine(minMaxProduct(arr1, arr2,
                                        n1, n2));
    }
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP program to find the
// to calculate the product
// of max element of first
// array and min element
// of second array

// Function to calculate
// the product
function minMaxProduct($arr1, $arr2,
                       $n1, $n2)
{

    // Initialize max of
    // first array
    $max = $arr1[0];
```

```
    // initialize min of
    // second array
    $min = $arr2[0];

    $i;
    for ($i = 1; $i < $n1 &&
                 $i < $n2; ++$i)
    {

        // To find the maximum
        // element in first array
        if ($arr1[$i] > $max)
            $max = $arr1[$i];

        // To find the minimum element
        // in second array
        if ($arr2[$i] < $min)
            $min = $arr2[$i];
    }

    // Process remaining elements
    while ($i < $n1)
    {
        if ($arr1[$i] > $max)
        $max = $arr1[$i];
        $i++;
    }
    while ($i < $n2)
    {
        if ($arr2[$i] < $min)
        $min = $arr2[$i];
        $i++;
    }

    return $max * $min;
}

    // Driven code
    $arr1 = array(10, 2, 3,
                  6, 4, 1);
    $arr2 = array(5, 1, 4,
                  2, 6, 9);
    $n1 = count($arr1);
    $n2 = count($arr2);
    echo minMaxProduct($arr1, $arr2,
                       $n1, $n2);

// This code is contributed by anuj_67.
```

```
?>
```

Output :

```
10
```

Time Complexity : O(n)
Space Complexity : O(1)

Improved By : shrikanth13, vt_m

Source

https://www.geeksforgeeks.org/product-maximum-first-array-minimum-second/

Chapter 210

Program to find largest element in an array

Program to find largest element in an array - GeeksforGeeks

Given an array, find the largest element in it.

Example:

```
Input : arr[] = {10, 20, 4}
Output : 20

Input : arr[] = {20, 10, 20, 4, 100}
Output : 100
```

The solution is to initialize max as first element, then traverse the given array from second element till end. For every traversed element, compare it with max, if it is greater than max, then update max.

C++

```
 // C++ program to find maximum
// in arr[] of size n
#include <bits/stdc++.h>
using namespace std;

int largest(int arr[], int n)
{
    int i;

    // Initialize maximum element
```

```
    int max = arr[0];

    // Traverse array elements
    // from second and compare
    // every element with current max
    for (i = 1; i < n; i++)
        if (arr[i] > max)
            max = arr[i];

    return max;
}

// Driver Code
int main()
{
    int arr[] = {10, 324, 45, 90, 9808};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Largest in given array is "
         << largest(arr, n);
    return 0;
}

// This Code is contributed
// by Shivi_Aggarwal
```

C

```
 // C program to find maximum in arr[] of size n
#include <stdio.h>

// C function to find maximum in arr[] of size n
int largest(int arr[], int n)
{
    int i;

    // Initialize maximum element
    int max = arr[0];

    // Traverse array elements from second and
    // compare every element with current max
    for (i = 1; i < n; i++)
        if (arr[i] > max)
            max = arr[i];

    return max;
}

int main()
```

```
{
    int arr[] = {10, 324, 45, 90, 9808};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Largest in given array is %d", largest(arr, n));
    return 0;
}
```

Java

```
 // Java Program to find maximum in arr[]
class Test
{
     static int arr[] = {10, 324, 45, 90, 9808};

     // Method to find maximum in arr[]
     static int largest()
     {
         int i;

         // Initialize maximum element
         int max = arr[0];

         // Traverse array elements from second and
         // compare every element with current max
         for (i = 1; i < arr.length; i++)
             if (arr[i] > max)
                 max = arr[i];

         return max;
     }

     // Driver method
     public static void main(String[] args)
     {
         System.out.println("Largest in given array is " + largest());
        }
 }
```

Python3

```
 # Python3 program to find maximum
# in arr[] of size n

# python function to find maximum
# in arr[] of size n
def largest(arr,n):
```

```
    # Initialize maximum element
    max = arr[0]

    # Traverse array elements from second
    # and compare every element with
    # current max
    for i in range(1, n):
        if arr[i] > max:
            max = arr[i]
    return max

# Driver Code
arr = [10, 324, 45, 90, 9808]
n = len(arr)
Ans = largest(arr,n)
print ("Largest in given array is",Ans)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# Program to find maximum in arr[]
using System;

class GFG {

    static int []arr = {10, 324, 45, 90, 9808};

    // Method to find maximum in arr[]
    static int largest()
    {
        int i;

        // Initialize maximum element
        int max = arr[0];

        // Traverse array elements from second and
        // compare every element with current max
        for (i = 1; i < arr.Length; i++)
            if (arr[i] > max)
                max = arr[i];

        return max;
    }

    // Driver method
    public static void Main()
    {
```

```
        Console.WriteLine("Largest in given "
                  + "array is " + largest());
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find maximum
// in arr[] of size n

// PHP function to find maximum
// in arr[] of size n
function largest($arr, $n)
{
    $i;

    // Initialize maximum element
    $max = $arr[0];

    // Traverse array elements
    // from second and
    // compare every element
    // with current max
    for ($i = 1; $i < $n; $i++)
        if ($arr[$i] > $max)
            $max = $arr[$i];

    return $max;
}

    // Driver Code
    $arr= array(10, 324, 45, 90, 9808);
    $n = sizeof($arr);
    echo "Largest in given array is "
                 , largest($arr, $n);

// This code is contributed by aj_36
?>
```

Output:

```
Largest in given array is 9808
```

Using Library Function :
We use std::max_element in C++.

C++

```
 // C++ program to find maximum in arr[] of size n
#include <bits/stdc++.h>
using namespace std;

// returns maximum in arr[] of size n
int largest(int arr[], int n)
{
    return *max_element(arr, arr+n);
}

int main()
{
    int arr[] = {10, 324, 45, 90, 9808};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << largest(arr, n);
    return 0;
}
```

Java

```
 // Java program to
// find maximum in
// arr[] of size n
import java .io.*;
import java.util.*;

class GFG
{

    // returns maximum in
    // arr[] of size n
    static int largest(int []arr,
                       int n)
    {
        Arrays.sort(arr);
        return arr[n - 1];
    }

    // Driver code
    static public void main (String[] args)
    {
        int []arr = {10, 324, 45,
                     90, 9808};
```

```
        int n = arr.length;
        System.out.println(largest(arr, n));
    }
}

// This code is contributed
// by anuj_67.
```

Python3

```
 # Python 3 program to find
# maximum in arr[] of size n

# returns maximum
# in arr[] of size n
def largest(arr, n):

    return max(arr)

# driver code
arr = [10, 324, 45, 90, 9808]
n = len(arr)

print(largest(arr, n))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
 // C# program to find maximum in
// arr[] of size n
using System;
using System.Linq;

public class GFG {

    // returns maximum in arr[] of size n
    static int largest(int []arr, int n)
    {
        return arr.Max();
    }

    // Driver code
    static public void Main ()
    {
        int []arr = {10, 324, 45, 90, 9808};
```

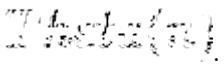
```
        int n = arr.Length;
        Console.WriteLine( largest(arr, n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// PHP program to find maximum
// in arr[] of size n

// returns maximum in
// arr[] of size n
function largest( $arr, $n)
{
    return max($arr);
}

    // Driver COde
    $arr = array(10, 324, 45, 90, 9808);
    $n = count($arr);
    echo largest($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
9808
```

Time complexity of the above solution is  .

Refer below article for more methods.
Program to find the minimum (or maximum) element of an array

Improved By : jit_t, vt_m, Shivi_Aggarwal

Source

https://www.geeksforgeeks.org/c-program-find-largest-element-array/

Chapter 211

Program to find the minimum (or maximum) element of an array

Program to find the minimum (or maximum) element of an array - GeeksforGeeks

Given an array, write functions to find minimum and maximum elements in it.

C++

```
 // CPP program to find minimum (or maximum) element
// in an array.
#include <iostream>
using namespace std;

int getMin(int arr[], int n)
{
    int res = arr[0];
    for (int i = 1; i < n; i++)
        res = min(res, arr[i]);
    return res;
}

int getMax(int arr[], int n)
{
    int res = arr[0];
    for (int i = 1; i < n; i++)
        res = max(res, arr[i]);
    return res;
}
```

```
int main()
{
    int arr[] = { 12, 1234, 45, 67, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Minimum element of array: " << getMin(arr, n) << "\n";
    cout << "Maximum element of array: " << getMax(arr, n);
    return 0;
}
```

Java

```
 // Java program to find minimum (or maximum)
// element in an array.
import java.io.*;

class GFG {

    static int getMin(int arr[], int n)
    {
        int res = arr[0];

        for (int i = 1; i < n; i++)
            res = Math.min(res, arr[i]);
        return res;
    }

    static int getMax(int arr[], int n)
    {
        int res = arr[0];

        for (int i = 1; i < n; i++)
            res = Math.max(res, arr[i]);
        return res;
    }

    // Driver code
    public static void main (String[] args)
    {
        int arr[] = { 12, 1234, 45, 67, 1 };
        int n = arr.length;
        System.out.println( "Minimum element"
           + " of array: " + getMin(arr, n));
        System.out.println( "Maximum element"
           + " of array: " + getMax(arr, n));
    }
}

// This code is contributed by anuj_67.
```

Python3

```
 # Python3 program to find minimum
# (or maximum) element in an array

# Minimum Function
def getMin(arr, n):
    res = arr[0]
    for i in range(1,n):
        res = min(res, arr[i])
    return res

# Maximum Function
def getMax(arr, n):
    res = arr[0]
    for i in range(1,n):
        res = max(res, arr[i])
    return res

# Driver Program
arr = [12, 1234, 45, 67, 1]
n = len(arr)
print ("Minimum element of array:", getMin(arr, n))
print ("Maximum element of array:", getMax(arr, n))

# This code is contributed
# by Shreyanshi Arun.
```

C#

```
 // C# program to find
// minimum (or maximum)
// element in an array.
using System;

class GFG
{
    static int getMin(int []arr,
                      int n)
    {
        int res = arr[0];

        for (int i = 1; i < n; i++)
            res = Math.Min(res, arr[i]);
        return res;
    }
```

```
    static int getMax(int []arr,
                      int n)
    {
        int res = arr[0];

        for (int i = 1; i < n; i++)
            res = Math.Max(res, arr[i]);
        return res;
    }

    // Driver code
    public static void Main ()
    {
        int []arr = {12, 1234, 45, 67, 1};
        int n = arr.Length;
        Console.Write("Minimum element" +
                         " of array: " +
                        getMin(arr, n) + "\n" );
        Console.Write("Maximum element" +
                         " of array: " +
                        getMax(arr, n));
    }
}

// This code is contributed by Smita.
```

PHP

```
 <?php
// PHP program to find minimum
// (or maximum) element in an
// array.

function getMin($arr, $n)
{
    $res = $arr[0];
    for ($i = 1; $i < $n; $i++)
        $res = min($res, $arr[$i]);
    return $res;
}

function getMax($arr, $n)
{
    $res = $arr[0];
    for ($i = 1; $i < $n; $i++)
        $res = max($res, $arr[$i]);
    return $res;
}
```

```
    // Driver Code
    $arr = array(12, 1234, 45, 67, 1);
    $n = sizeof($arr);
    echo "Minimum element of array: "
           , getMin($arr, $n), "\n";
    echo "Maximum element of array: "
                  ,getMax($arr, $n);

// This code is contributed by ajit
?>
```

Output:

```
Minimum element of array: 1
Maximum element of array: 1234
```

Time Complexity:O(n)

Recursive Solution

C++

```
 // CPP program to find minimum (or maximum) element
// in an array.
#include <iostream>
using namespace std;

int getMin(int arr[], int n)
{
    // If there is single element, return it.
    // Else return minimum of first element and
    // minimum of remaining array.
    return (n == 1) ? arr[0] : min(arr[0], getMin(arr + 1, n - 1));
}

int getMax(int arr[], int n)
{
    // If there is single element, return it.
    // Else return maximum of first element and
    // maximum of remaining array.
    return (n == 1) ? arr[0] : max(arr[0], getMax(arr + 1, n - 1));
}

int main()
{
    int arr[] = { 12, 1234, 45, 67, 1 };
```

```
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Minimum element of array: " << getMin(arr, n) << "\n";
    cout << "Maximum element of array: " << getMax(arr, n);
    return 0;
}
```

Output:

```
Min of array: 1
Max of array: 1234
```

Using Library functions:
We can use min_element() and max_element() to find minimum and maximum of array.

C++

```
 // CPP program to find minimum (or maximum) element
// in an array.
#include <iostream>
using namespace std;

int getMin(int arr[], int n)
{
    return *min_element(arr, arr + n);
}

int getMax(int arr[], int n)
{
    return *max_element(arr, arr + n);
}

int main()
{
    int arr[] = { 12, 1234, 45, 67, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Minimum element of array: " << getMin(arr, n) << "\n";
    cout << "Maximum element of array: " << getMax(arr, n);
    return 0;
}
```

Output:

```
Min of array: 1
Max of array: 1234
```

Improved By : jit_t, vt_m, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/program-find-minimum-maximum-element-array/

Chapter 212

Program to remove vowels from a String

Program to remove vowels from a String - GeeksforGeeks

Given a string, remove the vowels from the string and print the string without vowels.

Examples:

```
Input : welcome to geeksforgeeks
Output : wlcm t gksfrgks

Input : what is your name ?
Output : wht s yr nm ?
```

A loop is designed that goes through a list composed of the characters of that string, removes the vowels and then joins them.

Java

```
 // Java program to remove vowels from a String

import java.util.Arrays;
import java.util.List;

class Test
{
    static String remVowel(String str)
    {
         Character vowels[] = {'a', 'e', 'i', 'o', 'u','A','E','I','O','U'};
```

```
         List<Character> al = Arrays.asList(vowels);

         StringBuffer sb = new StringBuffer(str);

         for (int i = 0; i < sb.length(); i++) {

             if(al.contains(sb.charAt(i))){
                sb.replace(i, i+1, "") ;
                i--;
             }
        }


        return sb.toString();
    }
    // Driver method to test the above function
    public static void main(String[] args)
    {
        String str = "GeeeksforGeeks - A Computer Science Portal for Geeks";

        System.out.println(remVowel(str));
    }
}
```

Python

```
 # Python program to remove vowels from a string
# Function to remove vowels
def rem_vowel(string):
    vowels = ('a', 'e', 'i', 'o', 'u')
    for x in string.lower():
        if x in vowels:
            string = string.replace(x, "")

    # Print string without vowels
    print(string)

# Driver program
string = "GeeksforGeeks - A Computer Science Portal for Geeks"
rem_vowel(string)
```

Output:

```
GksfrGks - A Cmptr Scnc Prtl fr Gks
```

Source

https://www.geeksforgeeks.org/program-remove-vowels-string/

Chapter 213

Queries for greater than and not less than

Queries for greater than and not less than - GeeksforGeeks

Given an array of **N** integers. There will be **Q** queries, each include two integer of form **q** and **x**, 0 <= q <= 1. Queries are of two types:

- In first query (q = 0), the task is to find count of integers which are not less than x (OR greater than or equal to x).
- In second query (q = 1), the task is to find count of integers greater than x.

Examples:

```
Input : arr[] = { 1, 2, 3, 4 } and Q = 3
        Query 1: 0 5
        Query 2: 1 3
        Query 3: 0 3
Output :0
        1
        2
Explanation:
x = 5, q = 0 : There are no elements greater than or equal to it.
x = 3, q = 1 : There is one element greater than 3 which is 4.
x = 3, q = 0 : There are two elements greater than or equal to 3.
```

Method 1: A **Naive approach** can be for each query, traverse the whole array and count integers less or greater than x, depending on q. Time Complexity for this approach will be **O(Q*N)**.

Method 2: An **efficient** approach can be sort the array and use binary search for each query. This will take **O(NlogN + QlogN)**.

Below is the implementation of this approach :

C++

```
 // C++ to find number of integer less or greater given
// integer queries
#include<bits/stdc++.h>
using namespace std;

// Return the index of integer which are not less than x
// (or greater than or equal to x)
int lower_bound(int arr[], int start, int end, int x)
{
    while (start < end)
    {
        int mid = (start + end)>>1;
        if (arr[mid] >= x)
            end = mid;
        else
            start = mid + 1;
    }

    return start;
}

// Return the index of integer which are greater than x.
int upper_bound(int arr[], int start, int end, int x)
{
    while (start < end)
    {
        int mid = (start + end)>>1;
        if (arr[mid] <= x)
            start = mid + 1;
        else
            end = mid;
    }

    return start;
}

void query(int arr[], int n, int type, int x)
{
    // Counting number of integer which are greater than x.
    if (type)
        cout << n - upper_bound(arr, 0, n, x) << endl;

    // Counting number of integer which are not less than x
    // (Or greater tha or equal to x)
```

```
    else
        cout << n - lower_bound(arr, 0, n, x) << endl;
}

// Driven Program
int main()
{
    int arr[] = { 1, 2, 3, 4 };
    int n = sizeof(arr)/sizeof(arr[0]);

    sort(arr, arr + n);

    query(arr, n, 0, 5);
    query(arr, n, 1, 3);
    query(arr, n, 0, 3);

    return 0;
}
```

Java

```
 // Java to find number of integer
// less or greater given
// integer queries
import java.util.Arrays;
class GFG
{
// Return the index of integer
// which are not less than x
// (or greater than or equal
// to x)
static int lower_bound(int arr[], int start,
                            int end, int x)
{
    while (start < end)
    {
        int mid = (start + end)>>1;
        if (arr[mid] >= x)
            end = mid;
        else
            start = mid + 1;
    }

    return start;
}

// Return the index of integer
// which are greater than x.
```

```
static int upper_bound(int arr[], int start, int end, int x)
{
    while (start < end)
    {
        int mid = (start + end)>>1;
        if (arr[mid] <= x)
            start = mid + 1;
        else
            end = mid;
    }

    return start;
}

static void query(int arr[], int n, int type, int x)
{
    // Counting number of integer
    // which are greater than x.
    if (type==1)
        System.out.println(n - upper_bound(arr, 0, n, x));

    // Counting number of integer which
    // are not less than x (Or greater
    // than or equal to x)
    else
        System.out.println(n - lower_bound(arr, 0, n, x));
}

// Driver code
public static void main (String[] args)
{
    int arr[] = { 1, 2, 3, 4 };
    int n = arr.length;

    Arrays.sort(arr);

    query(arr, n, 0, 5);
    query(arr, n, 1, 3);
    query(arr, n, 0, 3);
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program to find number of integer
# less or greater given integer queries
```

```
# Return the index of integer
# which are not less than x
# (or greater than or equal to x)
def lower_bound(arr, start, end, x):

    while (start < end):

        mid = (start + end) >> 1
        if (arr[mid] >= x):
            end = mid
        else:
            start = mid + 1

    return start

# Return the index of integer
# which are greater than x.
def upper_bound(arr, start, end, x):

    while (start < end):

        mid = (start + end) >> 1
        if (arr[mid] <= x):
            start = mid + 1
        else:
            end = mid

    return start

def query(arr, n, type, x):

    # Counting number of integer
    # which are greater than x.
    if (type == 1):
        print(n - upper_bound(arr, 0, n, x))

    # Counting number of integer
    # which are not less than x
    # (Or greater tha or equal to x)
    else:
        print(n - lower_bound(arr, 0, n, x))

# Driver code
arr = [ 1, 2, 3, 4 ]
n =len(arr)

arr.sort()
```

```
query(arr, n, 0, 5)
query(arr, n, 1, 3)
query(arr, n, 0, 3)

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# to find number of integer less
// or greater given integer queries
using System;

class GFG {

// Return the index of integer which are
// not less than x (or greater than or
// equal to x)
static int lower_bound(int []arr, int start,
                            int end, int x)
{
    while (start < end)
    {
        int mid = (start + end)>>1;
        if (arr[mid] >= x)
            end = mid;
        else
            start = mid + 1;
    }

    return start;
}

// Return the index of integer
// which are greater than x.
static int upper_bound(int []arr, int start,
                            int end, int x)
{
    while (start < end)
    {
        int mid = (start + end)>>1;
        if (arr[mid] <= x)
            start = mid + 1;
        else
            end = mid;
    }

    return start;
```

```
}

static void query(int []arr, int n, int type, int x)
{
    // Counting number of integer
    // which are greater than x.
    if (type==1)
        Console.WriteLine(n - upper_bound(arr, 0, n, x));

    // Counting number of integer which
    // are not less than x (Or greater
    // than or equal to x)
    else
        Console.WriteLine(n - lower_bound(arr, 0, n, x));
}

// Driver code
public static void Main ()
{
    int []arr = {1, 2, 3, 4};
    int n = arr.Length;

    Array.Sort(arr);

    query(arr, n, 0, 5);
    query(arr, n, 1, 3);
    query(arr, n, 0, 3);
}
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP to find number of integer
// less or greater given
// integer queries

// Return the index of integer
// which are not less than x
// (or greater than or equal to x)
function lower_bound($arr, $start, $end, $x)
{
    while ($start < $end)
    {
    $mid = ($start + $end) >> 1;
        if ($arr[$mid] >= $x)
```

```
            $end = $mid;
        else
            $start = $mid + 1;
    }

    return $start;
}

// Return the index of integer
// which are greater than x.
function upper_bound($arr, $start, $end, $x)
{
    while ($start < $end)
    {
        $mid = ($start + $end) >> 1;
        if ($arr[$mid] <= $x)
            $start = $mid + 1;
        else
            $end = $mid;
    }

    return $start;
}

function query($arr, $n, $type, $x)
{

    // Counting number of integer
    // which are greater than x.
    if ($type)
        echo $n - upper_bound($arr, 0, $n, $x) ,"\n";

    // Counting number of integer
    // which are not less than x
    // (Or greater tha or equal to x)
    else
        echo $n - lower_bound($arr, 0, $n, $x) ,"\n";
}

    // Driver Code
    $arr = array(1, 2, 3, 4);
    $n = count($arr);

    sort($arr);

    query($arr, $n, 0, 5);
    query($arr, $n, 1, 3);
    query($arr, $n, 0, 3);
```

```
// This code is contributed by anuj_67.
?>
```

Output:

```
0
1
2
```

Time Complexity : O((N + Q) * logN).

Improved By : vt_m

Source

https://www.geeksforgeeks.org/queries-greater-not-less/

Chapter 214

Quickselect Algorithm

Quickselect Algorithm - GeeksforGeeks

Quickselect is a selection algorithm to find the k-th smallest element in an unordered list. It is related to the quick sort sorting algorithm.

Examples:

```
Input: arr[] = {7, 10, 4, 3, 20, 15}
           k = 3
Output: 7

Input: arr[] = {7, 10, 4, 3, 20, 15}
           k = 4
Output: 10
```

The algorithm is similar to QuickSort. The difference is, instead of recurring for both sides (after finding pivot), it recurs only for the part that contains the k-th smallest element. The logic is simple, if index of partitioned element is more than k, then we recur for left part. If index is same as k, we have found the k-th smallest element and we return. If index is less than k, then we recur for right part. This reduces the expected complexity from O(n log n) to O(n), with a worst case of O(n^2).

```
function quickSelect(list, left, right, k)

   if left = right
      return list[left]

   Select a pivotIndex between left and right

   pivotIndex := partition(list, left, right,
```

```
                                    pivotIndex)
   if k = pivotIndex
      return list[k]
   else if k < pivotIndex
      right := pivotIndex - 1
   else
      left := pivotIndex + 1
```

```
 // CPP program for implementation of QuickSelect
#include <bits/stdc++.h>
using namespace std;

// Standard partition process of QuickSort().
// It considers the last element as pivot
// and moves all smaller element to left of
// it and greater elements to right
int partition(int arr[], int l, int r)
{
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++) {
        if (arr[j] <= x) {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}

// This function returns k'th smallest
// element in arr[l..r] using QuickSort
// based method.  ASSUMPTION: ALL ELEMENTS
// IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
    // If k is smaller than number of
    // elements in array
    if (k > 0 && k <= r - l + 1) {

        // Partition the array around last
        // element and get position of pivot
        // element in sorted array
        int index = partition(arr, l, r);

        // If position is same as k
        if (index - l == k - 1)
            return arr[index];
```

```
        // If position is more, recur
        // for left subarray
        if (index - l > k - 1)
            return kthSmallest(arr, l, index - 1, k);

        // Else recur for right subarray
        return kthSmallest(arr, index + 1, r,
                           k - index + l - 1);
    }

    // If k is more than number of
    // elements in array
    return INT_MAX;
}

// Driver program to test above methods
int main()
{
    int arr[] = { 10, 4, 5, 8, 6, 11, 26 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    cout << "K-th smallest element is "
        << kthSmallest(arr, 0, n - 1, k);
    return 0;
}
```

Output:

```
K-th smallest element is 6
```

Important Points:

1. Like quicksort, it is fast in practice, but has poor worst-case performance. It is used in
2. The partition process is same as QuickSort, only recursive code differs.
3. There exists an algorithm that finds k-th smallest element in O(n) in worst case, but QuickSelect performs better on average.

Related C++ function : std::nth_element in C++

Source

https://www.geeksforgeeks.org/quickselect-algorithm/

Chapter 215

Randomized Binary Search Algorithm

Randomized Binary Search Algorithm - GeeksforGeeks

We are given a sorted array A[] of n elements. We need to find if x is present in A or not.In binary search we always used middle element, here we will randomly pick one element in given range.

In Binary Search we had

```
middle = (start + end)/2
```

In Randomized binary search we do following

```
Generate a random number t
Since range of number in which we want a random
number is [start, end]
Hence we do, t = t % (end-start+1)
Then, t = start + t;
Hence t is a random number between start and end
```

It is a Las Vegas randomized algorithm as it always finds the correct result.

Expected Time complexity of Randomized Binary Search Algorithm
For n elements let say expected time required be T(n), After we choose one random pivot, array size reduces to say k. Since pivot is chosen with equal probability for all possible pivots, hence p = 1/n.

T(n) is sum of time of all possible sizes after choosing pivot multiplied by probability of choosing that pivot plus time take to generate random pivot index.Hence

```
T(n) = p*T(1) + p*T(2) + ..... + p*T(n) + 1
putting p = 1/n
T(n) = ( T(1) + T(2) + ..... + T(n) ) / n + 1
n*T(n) = T(1) + T(2) + .... + T(n) + n      .... eq(1)
Similarly for n-1
(n-1)*T(n-1) = T(1) + T(2) + ..... + T(n-1) + n-1    .... eq(2)
Subtract eq(1) - eq(2)
n*T(n) - (n-1)*T(n-1) = T(n) + 1
(n-1)*T(n) - (n-1)*T(n-1) =  1
(n-1)*T(n) = (n-1)*T(n-1) + 1
T(n) = 1/(n-1) + T(n-1)
T(n) = 1/(n-1) + 1/(n-2) + T(n-2)
T(n) = 1/(n-1) + 1/(n-2) + 1/(n-3) + T(n-3)
Similarly,
T(n) = 1 + 1/2 + 1/3 + ... + 1/(n-1)
Hence T(n) is equal to (n-1)th Harmonic number,
n-th harmonic number is O(log n)
Hence T(n) is O(log n)
```

Recursive C++ implementation of Randomized Binary Search

```
 // C++ program to implement recursive
// randomized algorithm.
#include <iostream>
#include <ctime>
using namespace std;

// To generate random number
// between x and y ie.. [x, y]
int getRandom(int x, int y)
{
    srand(time(NULL));
    return (x + rand() % (y-x+1));
}

// A recursive randomized binary search function.
// It returns location of x in
// given array arr[l..r] is present, otherwise -1
int randomizedBinarySearch(int arr[], int l,
                           int r, int x)
{
    if (r >= l)
    {
        // Here we have defined middle as
        // random index between l and r ie.. [l, r]
        int mid = getRandom(l, r);
```

```
        // If the element is present at the
        // middle itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
          return randomizedBinarySearch(arr, l,
                                    mid-1, x);

        // Else the element can only be present
        // in right subarray
        return randomizedBinarySearch(arr, mid+1,
                                        r, x);
    }

    // We reach here when element is not present
    // in array
    return -1;
}

// Driver code
int main(void)
{
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr)/ sizeof(arr[0]);
    int x = 10;
    int result = randomizedBinarySearch(arr, 0, n-1, x);
    (result == -1)? printf("Element is not present in array")
    : printf("Element is present at index %d", result);
    return 0;
}
```

Output:

```
Element is present at index 3
```

Iterative C++ implementation of Randomized Binary Search

```
 // C++ program to implement iterative
// randomized algorithm.
#include <iostream>
#include <ctime>
using namespace std;
```

```
// To generate random number
// between x and y ie.. [x, y]
int getRandom(int x, int y)
{
    srand(time(NULL));
    return (x + rand()%(y-x+1));
}

// A iterative randomized binary search function.
// It returns location of x in
// given array arr[l..r] if present, otherwise -1
int randomizedBinarySearch(int arr[], int l,
                                int r, int x)
{
    while (l <= r)
    {
        // Here we have defined middle as
        // random index between l and r ie.. [l, r]
        int m = getRandom(l, r);

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }

    // if we reach here, then element was
    // not present
    return -1;
}

// Driver code
int main(void)
{
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr)/ sizeof(arr[0]);
    int x = 10;
    int result = randomizedBinarySearch(arr, 0, n-1, x);
    (result == -1)? printf("Element is not present in array")
        : printf("Element is present at index %d", result);
    return 0;
```

```
}
```

Output:

```
Element is present at index 3
```

Source

https://www.geeksforgeeks.org/randomized-binary-search-algorithm/

Chapter 216

Reallocation of elements based on Locality of Reference

Reallocation of elements based on Locality of Reference - GeeksforGeeks

Consider a problem where same elements are likely to be searched again and again. Implement search operation efficiently.

Examples :

```
Input : arr[] = {12 25 36 85 98 75 89 15 63 66
                                64 74 27 83 97}
          q[] = {63, 63, 86, 63, 78}
Output : Yes Yes No Yes No
We need one by one search items of q[] in arr[].
The element 63 is present, 78 and 86 are not present.
```

The idea is **simple**, we move the searched element to front of the array so that it can be searched quickly next time.

```
 // C++ program to implement search for an item
// that is searched again and again.
#include <bits/stdc++.h>
using namespace std;

// A function to perform sequential search.
bool search(int arr[], int n, int x)
{
    // Linearly search the element
    int res = -1;
    for (int i = 0; i < n; i++)
```

```
        if (x == arr[i])
           res = i;

    // If not found
    if (res == -1)
        return false;

    // Shift elements before one position
    int temp = arr[res];
    for (int i = res; i > 0; i--)
        arr[i] = arr[i - 1];

    arr[0] = temp;
    return true;
}

// Driver Code
int main()
{
    int arr[] = { 12, 25, 36, 85, 98, 75, 89, 15,
                  63, 66, 64, 74, 27, 83, 97 };
    int q[] = {63, 63, 86, 63, 78};
    int n = size(arr)/sizeof(arr[0]);
    int m = sizeof(q)/sizeof(q[0]);
    for (int i=0; i<m; i++)
      search(arr, n, q[i]?  cout << "Yes "
                         :  cout << "No  ";

    return 0;
}
```

Further Thoughts : We can do better by using a linked list. In linked list, moving an item to front can be done in O(1) time.
The best solution would be to use Splay Tree (a data structure designed for this purpose). Splay tree supports insert, search and delete operations in O(Log n) time on average. Also, splay tree is a BST, so we can quickly print elements in sorted order.

Source

https://www.geeksforgeeks.org/reallocation-of-elements-based-on-locality-of-reference/

Chapter 217

Recursive program to linearly search an element in a given array

Recursive program to linearly search an element in a given array - GeeksforGeeks

Given an unsorted array and an element x, search x in given array. Write recursive C code for this. If element is not present, return -1.

Approach : The idea is to compare x with first element in arr[]. If element is found at first position, return it. Else recur for remaining array and x.

C++

```
 // Recursive C++ program
// to search x in array
#include<bits/stdc++.h>

using namespace std;

// Recursive function to
// search x in arr[l..r]
int recSearch(int arr[], int l,
              int r, int x)
{
    if (r < l)
        return -1;
    if (arr[l] == x)
        return l;
    if (arr[r] == x)
        return r;
```

```
    return recSearch(arr, l + 1,
                          r - 1, x);
}

// Driver Code
int main()
{
    int arr[] = {12, 34, 54, 2, 3}, i;
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 3;
    int index = recSearch(arr, 0, n - 1, x);
    if (index != -1)
    cout << "Element " << x
         << " is present at index "
         << index;
    else
        cout << "Element" << x
             << " is not present" ;
    return 0;
}

// This code is contributed
// by Shivi_Aggarwal
```

C

```
 #include<stdio.h>

/* Recursive function to search x in arr[l..r] */
int recSearch(int arr[], int l, int r, int x)
{
     if (r < l)
        return -1;
     if (arr[l] == x)
        return l;
     if (arr[r] == x)
        return r;
     return recSearch(arr, l+1, r-1, x);
}

int main()
{
    int arr[] = {12, 34, 54, 2, 3}, i;
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 3;
    int index = recSearch(arr, 0, n-1, x);
    if (index != -1)
       printf("Element %d is present at index %d", x, index);
```

```
    else
        printf("Element %d is not present", x);
    return 0;
}
```

Java

```
 // Recursive Java program to search x in array
class Test
{
     static int arr[] = {12, 34, 54, 2, 3};

     /* Recursive Method to search x in arr[l..r] */
     static int recSearch(int arr[], int l, int r, int x)
     {
          if (r < l)
             return -1;
          if (arr[l] == x)
             return l;
          if (arr[r] == x)
             return r;
          return recSearch(arr, l+1, r-1, x);
     }

     // Driver method
     public static void main(String[] args)
     {
        int x = 3;

        //Method call to find x
        int index = recSearch(arr, 0, arr.length-1, x);
        if (index != -1)
           System.out.println("Element " + x + " is present at index " +
                                                    index);
        else
            System.out.println("Element " + x + " is not present");
        }
 }
```

Python

```
 # Recursive function to search x in arr[l..r]
def recSearch( arr, l, r, x):
    if r < l:
        return -1
    if arr[l] == x:
        return l
```

```
    if arr[r] == x:
        return r
    return recSearch(arr, l+1, r-1, x)

# Driver Code
arr = [12, 34, 54, 2, 3]
n = len(arr)
x = 3
index = recSearch(arr, 0, n-1, x)
if index != -1:
    print "Element", x,"is present at index %d" %(index)
else:
    print "Element %d is not present" %(x)

# Contributed By Harshit Agrawal
```

C#

```
 // Recursive C# program to
// search x in array
using System;

static class Test
{
    static int []arr = {12, 34, 54, 2, 3};

    /* Recursive Method to search x in arr[l..r] */
    static int recSearch(int []arr, int l, int r, int x)
    {
        if (r < l)
            return -1;
        if (arr[l] == x)
            return l;
        if (arr[r] == x)
            return r;
        return recSearch(arr, l+1, r-1, x);
    }

    // Driver method
    public static void Main(String[] args)
    {
        int x = 3;

        // Method call to find x
        int index = recSearch(arr, 0, arr.Length-1, x);

        if (index != -1)
        Console.Write("Element " + x +
```

```
                    " is present at index " + index);
        else
            Console.Write("Element " + x +
                          " is not present");
        }
}

// This code is contributed by Smitha Dinesh Semwal
```

PHP

```
 <?php
// Recursive PHP program to
// search x in array

// Recursive function to
// search x in arr[l..r]
function recSearch($arr, int $l,
                   int $r, int $x)
{
    if ($r < $l)
        return -1;
    if ($arr[$l] == $x)
        return $l;
    if ($arr[$r] == $x)
        return $r;
     return recSearch($arr, $l+1, $r-1, $x);
}

    // Driver Code
    $arr = array(12, 34, 54, 2, 3); $i;
    $n = count($arr);
    $x = 3;
    $index = recSearch($arr, 0, $n - 1, $x);
    if ($index != -1)
    echo "Element"," ", $x," ",
         "is present at index ", $index;
    else
        echo "Element is not present", $x;

// This code is contributed by anuj_67.
?>
```

Output:

```
Element 3 is present at index 4
```

Improved By : Smitha Dinesh Semwal, vt_m, Shivi_Aggarwal, VineetGupta1

Source

https://www.geeksforgeeks.org/recursive-c-program-linearly-search-element-given-array/

Chapter 218

Recursive Programs to find Minimum and Maximum elements of array

Recursive Programs to find Minimum and Maximum elements of array - GeeksforGeeks

Given an array of integers, we need to find the minimum element of that array using recursion.

Examples :

```
Input : A = {1, 4, 3, -5, -4, 8, 6};
Output : -5

Input : A = {1, 4, 45, 6, 10, -8}
Output : -8
```

Program to find largest element in an array

Recursive Minimum

```
If there is single element, return it.
Else return minimum of following.
    a) Last Element
    b) Value returned by recursive call
       fir n-1 elements.
```

C++

```
 // Recursive C++ program to find minimum
#include <iostream>
using namespace std;

// function to print Minimum element using recursion
int findMinRec(int A[], int n)
{
    // if size = 0 means whole array has been traversed
    if (n == 1)
        return A[0];
    return min(A[n-1], findMinRec(A, n-1));
}

// driver code to test above function
int main()
{
    int A[] = {1, 4, 45, 6, -50, 10, 2};
    int n = sizeof(A)/sizeof(A[0]);
    cout <<  findMinRec(A, n);
    return 0;
}
```

java

```
 // Recursive Java program to find minimum
import java.util.*;

class GFG {

    // function to return minimum element using recursion
    public static int findMinRec(int A[], int n)
    {
      // if size = 0 means whole array
      // has been traversed
      if(n == 1)
        return A[0];

        return Math.min(A[n-1], findMinRec(A, n-1));
    }

    // Driver code
    public static void main(String args[])
    {
        int A[] = {1, 4, 45, 6, -50, 10, 2};
        int n = A.length;

        // Function calling
        System.out.println(findMinRec(A, n));
```

```
    }
}

//This code is contributed by Niraj_Pandey
```

PHP

```
 <?php
// Recursive PHP program to find minimum

// function to print Minimum
// element using recursion
function findMinRec($A, $n)
{

    // if size = 0 means whole
    // array has been traversed
    if ($n == 1)
        return $A[0];
    return min($A[$n - 1], findMinRec($A, $n - 1));
}

    // Driver Code
    $A = array (1, 4, 45, 6, -50, 10, 2);
    $n = sizeof($A);
    echo findMinRec($A, $n);

// This code is contributed by akt
?>
```

Output:

```
-50
```

Recursive Maximum

```
If there is single element, return it.
Else return maximum of following.
    a) Last Element
    b) Value returned by recursive call
       fir n-1 elements.
```

C++

```
 // Recursive C++ program to find maximum
#include <iostream>
using namespace std;

// function to return maximum element using recursion
int findMaxRec(int A[], int n)
{
    // if n = 0 means whole array has been traversed
    if (n == 1)
        return A[0];
    return max(A[n-1], findMaxRec(A, n-1));
}

// driver code to test above function
int main()
{
    int A[] = {1, 4, 45, 6, -50, 10, 2};
    int n = sizeof(A)/sizeof(A[0]);
    cout <<  findMinRec(A, n);
    return 0;
}
```

Java

```
 // Recursive Java program to find maximum
import java.util.*;

class GFG {

    // function to return maximum element using recursion
    public static int findMaxRec(int A[], int n)
    {
      // if size = 0 means whole array
      // has been traversed
      if(n == 1)
        return A[0];

        return Math.max(A[n-1], findMaxRec(A, n-1));
    }

    // Driver code
    public static void main(String args[])
    {
        int A[] = {1, 4, 45, 6, -50, 10, 2};
        int n = A.length;

        // Function calling
        System.out.println(findMaxRec(A, n));
```

```
    }
}

//This code is contributed by Niraj_Pandey
```

Output:

```
45
```

Improved By : Niraj_Pandey, jit_t

Source

https://www.geeksforgeeks.org/recursive-programs-to-find-minimum-and-maximum-elements-of-array/

Chapter 219

Recursive function to do substring search

Recursive function to do substring search - GeeksforGeeks

Given a text txt[] and a pattern pat[], write a recursive function "contains(char pat[], char txt[])" that returns true if pat[] is present in txt[], otherwise false.

Examples:

```
1) Input:   txt[] =  "THIS IS A TEST TEXT"
            pat[] = "TEST"
  Output:  true


2) Input:  txt[] =  "geeksforgeeks"
           pat[] = "quiz"
  Output:  false;
```

We strongly recommend to minimize the browser and try this yourself first.

Below is recursive algorithm.

```
contains(tex[], pat[])
    1) If current character is last character of text, but pat
       has more characters, return false.

    2) Else If current character is last character of pattern,
       then return true

    3) Else If current characters of pat and text match, then
```

```
        return contains(text + 1, pat + 1);

    4) Else If current characters of pat and text don't match
        return contains(text + 1, pat);
```

Below is C++ implementation of above algorithm.

```
 // Recursive C++ program to find if a given pattern is
// present in a text
#include<iostream>
using namespace std;

bool exactMatch(char *text, char *pat)
{
    if (*text == '\0' && *pat != '\0')
        return false;

    // Else If last character of pattern reaches
    if (*pat == '\0')
        return true;

    if (*text == *pat)
        return exactMatch(text + 1, pat + 1);

    return false;
}

// This function returns true if 'text' contain 'pat'
bool contains(char *text, char *pat)
{
    // If last character of text reaches
    if (*text == '\0')
        return false;

    // If current characts of pat and text match
    if (*text == *pat)
        if(exactMatch(text, pat))
            return 1;
        else
          return contains(text + 1, pat);

    // If current characts of pat and tex don't match
    return contains(text + 1, pat);
}

// Driver program to test above function
int main()
{
```

```
    cout << contains("geeksforgeeks", "geeks") << endl;
    cout << contains("geeksforgeeks", "geeksquiz") << endl;
    cout << contains("geeksquizgeeks", "quiz") << endl;
    return 0;
}
```

Output:

```
1
0
1
```

This article is contributeed by Bhupinder. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : san123

Source

https://www.geeksforgeeks.org/recursive-function-to-do-substring-search/

Chapter 220

Remove all occurrences of a character in a string

Remove all occurrences of a character in a string - GeeksforGeeks

Given a string. Write a program to remove all the occurrences of a character in the string.

Examples:

```
Input : s = "geeksforgeeks"
        c = 'e'
Output : s = "gksforgks"
```

```
Input : s = "geeksforgeeks"
        c = 'g'
Output : s = "eeksforeeks"
```

The idea is to maintain an index of resultant string.

C++

```
 // C++ program to remove a particular character
// from a string.
#include <bits/stdc++.h>
using namespace std;

void removeChar(char *s, int c){

    int j, n = strlen(s);
    for (int i=j=0; i<n; i++)
       if (s[i] != c)
```

```
          s[j++] = s[i];

    s[j] = '\0';
}

int main()
{
   char s[] = "geeksforgeeks";
   removeChar(s, 'g');
   cout << s;
   return 0;
}
```

Java

```
 // Java program to remove
// a particular character
// from a string.
class GFG
{
static void removeChar(String s, char c)
{
    int j, count = 0, n = s.length();
    char []t = s.toCharArray();
    for (int i = j = 0; i < n; i++)
    {
        if (t[i] != c)
        t[j++] = t[i];
        else
            count++;
    }

    while(count > 0)
    {
        t[j++] = '\0';
        count--;
    }

    System.out.println(t);
}

// Driver Code
public static void main(String[] args)
{
    String s = "geeksforgeeks";
    removeChar(s, 'g');
}
}
```

```
// This code is contributed
// by ChitraNayal
```

Python3

```
 # Python3 program to remove
# a particular character
# from a string.

# function for removing the
# occurence of character
def removeChar(s, c) :

    # find total no. of
    # occurence of charcter
    counts = s.count(c)

    # convert into list
    # of characters
    s = list(s)

    # keeep looping untill
    # counts become 0
    while counts :

        # remove character
        # from the list
        s.remove(c)

        # decremented by one
        counts -= 1

    # join all remaining characters
    # of the list with empty string
    s = '' . join(s)

    print(s)

# Driver code
if __name__ == '__main__' :

    s = "geeksforgeeks"
    removeChar(s,'g')

# This code is contributed
# by Ankit Rai
```

C#

```
 // C# program to remove a
// particular character
// from a string.
using System;

class GFG
{
static void removeChar(string s,
                       char c)
{
    int j, count = 0, n = s.Length;
    char[] t = s.ToCharArray();
    for (int i = j = 0; i < n; i++)
    {
        if (s[i] != c)
        t[j++] = s[i];
        else
            count++;
    }

    while(count > 0)
    {
        t[j++] = '\0';
        count--;
    }

    Console.Write(t);
}

// Driver Code
public static void Main()
{
    string s = "geeksforgeeks";
    removeChar(s, 'g');
}
}

// This code is contributed
// by ChitraNayal
```

PHP

```
 <?php
// PHP program to remove a
// particular character
```

```
// from a string.

function removeChar($s, $c)
{

    $n = strlen($s);
    $count = 0;
    for ($i = $j = 0; $i < $n; $i++)
    {
    if ($s[$i] != $c)
        $s[$j++] = $s[$i];
    else
        $count++;
    }
    while($count--)
    {
        $s[$j++] = NULL;
    }
    echo $s;
}

// Driver code
$s = "geeksforgeeks";
removeChar($s, 'g');

// This code is contributed
// by ChitraNayal
?>
```

Output:

```
eeksforeeks
```

Time Complexity : O(n) where n is length of input string.
Auxiliary Space : O(1)

Improved By : ANKITRAI1, ChitraNayal

Source

https://www.geeksforgeeks.org/remove-all-occurrences-of-a-character-in-a-string/

Chapter 221

Repeatedly search an element by doubling it after every successful search

Repeatedly search an element by doubling it after every successful search - GeeksforGeeks

Given an array "a[]" and integer "b". Find whether b is present in a[] or not. If present, then double the value of b and search again. We repeat these steps until b is not found. Finally we return value of b.

Examples:

```
Input : a[] = {1, 2, 3}
          b = 1
Output :4
Initially we start with b = 1. Since
it is present in array, it becomes 2.
Now 2 is also present in array b becomes 4 .
Since 4 is not present, we return 4.

Input : a[] = {1 3 5 2 12}
          b = 3
Output :6
```

Question Source : Asked in Yatra.com Online Test

1) Sort the input array.
2) Keep doing binary search and doubling until the element is not present.

The below code using binary_search() in STL

C++

```
 // C++ program to repeatedly search an element by
// doubling it after every successful search
#include <bits/stdc++.h>
using namespace std;

int findElement(int a[], int n, int b)
{
    // Sort the given array so that binary search
    // can be applied on it
    sort(a, a + n);

    int max = a[n - 1]; // Maximum array element

    while (b < max) {

        // search for the element b present or
        // not in array
        if (binary_search(a, a + n, b))
            b *= 2;
        else
            return b;
    }

    return b;
}

// Driver code
int main()
{
    int a[] = { 1, 2, 3 };
    int n = sizeof(a) / sizeof(a[0]);
    int b = 1;
    cout << findElement(a, n, b);
    return 0;
}
```

Java

```
 // Java program to repeatedly search an element by
// doubling it after every successful search
import java.util.Arrays;
public class Test4 {

    static int findElement(int a[], int n, int b)
    {
        // Sort the given array so that binary search
        // can be applied on it
        Arrays.sort(a);
```

```
        int max = a[n - 1]; // Maximum array element

        while (b < max) {

            // search for the element b present or
            // not in array
            if (Arrays.binarySearch(a, b) > -1)
                b *= 2;
            else
                return b;
        }

        return b;
    }

    // Driver code
    public static void main(String args[])
    {
        int a[] = { 1, 2, 3 };
        int n = a.length;
        int b = 1;
        System.out.println(findElement(a, n, b));
    }
}
// This article is contributed by Sumit Ghosh
```

Python

```
 # Python program to repeatedly search an element by
# doubling it after every successful search

def binary_search(a, x, lo=0, hi=None):
    if hi is None:
        hi = len(a)
    while lo < hi:
        mid = (lo+hi)//2
        midval = a[mid]
        if midval < x:
            lo = mid+1
        elif midval > x:
            hi = mid
        else:
            return mid
    return -1

def findElement(a, n, b):
```

```
    # Sort the given array so that binary search
    # can be applied on it
    a.sort()

    mx = a[n - 1] # Maximum array element

    while (b < max):

        # search for the element b present or
        # not in array
        if (binary_search(a, b, 0, n) != -1):
            b *= 2
        else:
            return b
    return b

# Driver code
a = [ 1, 2, 3 ]
n = len(a)
b = 1
print findElement(a, n, b)

# This code is contributed by Sachin Bisht
```

C#

```
 // C# program to repeatedly search an
// element by doubling it after every
// successful search
using System;

public class GFG {

    static int findElement(int []a,
                           int n, int b)
    {

        // Sort the given array so that
        // binary search can be applied
        // on it
        Array.Sort(a);

        // Maximum array element
        int max = a[n - 1];

        while (b < max) {

            // search for the element b
```

```
            // present or not in array
            if (Array.BinarySearch(a, b) > -1)
                b *= 2;
            else
                return b;
        }

        return b;
    }

    // Driver code
    public static void Main()
    {
        int []a = { 1, 2, 3 };
        int n = a.Length;
        int b = 1;
        Console.WriteLine(findElement(a, n, b));
    }
}

// This code is contributed by vt_m.
```

Output:

```
4
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/repeatedly-search-element-doubling-every-successful-search/

Chapter 222

Replace two consecutive equal values with one greater

Replace two consecutive equal values with one greater - GeeksforGeeks

You are given an array of size 'n'. You have to replace every pair of consecutive values 'x' by a single value 'x+1' every time until there is no such repetition left and then print the new array.

> Input : 5, 2, 1, 1, 2, 2
> Output : 5 4
> **Explanation:**
> step 1: While traversing, encountered pair of 1(gets replaced by 2. We get 5, 2, 2, 2, 2
> step 2: The first encountered pair of 2 gets replaced by 3. We get 5, 3, 2, 2
> step 3: Again pair of 2 gets replaced by 3. We get 5, 3, 3
> step 4: Recently formed pair of 3 gets replaced by 4. We get 5, 4
> This is our required answer.
>
> Input : 4, 5, 11, 2, 5, 7, 2
> Output : 4 5 11 2 5 7 2

Approach : In this problem you have to traverse the array of integers and check if any two consecutive integers are of a same value X. Then you have to replace that pair of integers with a single integer X+1. After that you have to begin with a new step by re-traversing the array and performing the same operation.

C++

```
 // C++ program to replace two elements with equal
// values with one greater.
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to replace consecutive equal
// elements
void replace_elements(int arr[], int n)
{
    int pos = 0;  // Index in result

    for (int i = 0; i < n; i++) {
        arr[pos++] = arr[i];

        while (pos > 1 && arr[pos - 2] ==
                          arr[pos - 1]) {
            pos--;
            arr[pos - 1]++;
        }
    }

    // to print new array
    for (int i = 0; i < pos; i++)
        cout << arr[i] << " ";
}

// Driver Code
int main()
{
    int arr[] = { 6, 4, 3, 4, 3, 3, 5 };
    int n = sizeof(arr) / sizeof(int);
    replace_elements(arr, n);
    return 0;
}
```

Java

```
 // java program to replace two elements
// with equal values with one greater.
public class GFG {

    // Function to replace consecutive equal
    // elements
    static void replace_elements(int arr[], int n)
    {
        int pos = 0; // Index in result

        for (int i = 0; i < n; i++) {
            arr[pos++] = arr[i];

            while (pos > 1 && arr[pos - 2] ==
                                arr[pos - 1])
```

```
            {
                pos--;
                arr[pos - 1]++;
            }
        }

        // to print new array
        for (int i = 0; i < pos; i++)
            System.out.print( arr[i] + " ");
    }

    // Driver code
    public static void main(String args[])
    {
        int arr[] = { 6, 4, 3, 4, 3, 3, 5 };
        int n = arr.length;
        replace_elements(arr, n);
    }
}

// This code is contributed by Sam007
```

Python3

```
 # python program to replace two elements
# with equal values with one greater.
from __future__ import print_function

# Function to replace consecutive equal
# elements
def replace_elements(arr, n):

    pos = 0 # Index in result

    for i in range(0, n):
        arr[pos] = arr[i]
        pos = pos + 1
        while (pos > 1 and arr[pos - 2]
                        == arr[pos - 1]):
            pos -= 1
            arr[pos - 1] += 1

    # to print new array
    for i in range(0, pos):
        print(arr[i], end=" ")

# Driver Code
arr = [ 6, 4, 3, 4, 3, 3, 5 ]
```

```
n = len(arr)
replace_elements(arr, n)

# This code is contributed by Sam007
```

C#

```
 // C# program to replace two elements
// with equal values with one greater.
using System;

class GFG {

    // Function to replace consecutive equal
    // elements
    static void replace_elements(int []arr, int n)
    {
        int pos = 0; // Index in result

        for (int i = 0; i < n; i++) {
            arr[pos++] = arr[i];

            while (pos > 1 && arr[pos - 2] ==
                                  arr[pos - 1])
            {
                pos--;
                arr[pos - 1]++;
            }
        }

        // to print new array
        for (int i = 0; i < pos; i++)
            Console.Write( arr[i] + " ");
    }

    // Driver code
    static void Main()
    {
        int []arr = { 6, 4, 3, 4, 3, 3, 5 };
        int n = arr.Length;
        replace_elements(arr, n);
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to replace two
// elements with equal
// values with one greater.

// Function to replace consecutive
// equal elements
function replace_elements($arr,$n)
{
    // Index in result
    $pos = 0;

    for ($i = 0; $i < $n; $i++)
    {
        $arr[$pos++] = $arr[$i];

        while ($pos > 1 && $arr[$pos - 2] ==
                        $arr[$pos - 1])
        {
            $pos--;
            $arr[$pos - 1]++;
        }
    }

    // to print new array
    for ($i = 0; $i < $pos; $i++)
        echo $arr[$i] . " ";
}

    // Driver Code
    $arr = array(6, 4, 3, 4, 3, 3, 5);
    $n = count($arr);
    replace_elements($arr, $n);

// This code is contributed by Sam007.
?>
```

Output:

```
6 4 3 6
```

Improved By : Sam007

Source

https://www.geeksforgeeks.org/replace-two-consecutive-equal-values-one-greater/

Chapter 223

Replacing an element makes array elements consecutive

Replacing an element makes array elements consecutive - GeeksforGeeks

Given an array of positive distinct integers. We need to find the only element whose replacement with any other value makes array elements distinct consecutive. If it is not possible to make array elements consecutive, return -1.

Examples :

```
Input : arr[] = {45, 42, 46, 48, 47}
Output : 42
Explanation: We can replace 42 with either
44 or 48 to make array consecutive.

Input : arr[] = {5, 6, 7, 9, 10}
Output : 5 [OR 10]
Explanation: We can either replace 5 with 8
or 10 with 8 to make array elements
consecutive.

Input : arr[] = {5, 6, 7, 9, 8}
Output : Array elements are already consecutive
```

A **Naive Approach** is to check each element of arr[], after replacing of which makes consecutive or not. Time complexity for this approach $O(n^2)$

A **Better Approach** is based on an important observation that either the smallest or the largest element would be answer if answer exists. If answer exists, then there are two cases. 1) Series of consecutive elements starts with minimum element of array then continues by adding 1 to previous.

2) Series of consecutive elements start with maximum element of array, then continues by subtracting 1 from previous.

We make above two series and for every series, we search series elements in array. If for both series, number of mismatches are more than 1, then answer does not exist. If any series is found with one mismatch, then we have answer.

C++

```
 // CPP program to find an element replacement
// of which makes the array elements consecutive.
#include <bits/stdc++.h>
using namespace std;

int findElement(int arr[], int n)
{
    sort(arr, arr+n);

    // Making a series starting from first element
    // and adding 1 to every element.
    int mismatch_count1 = 0, res;
    int next_element = arr[n-1] - n + 1;
    for (int i=0; i<n-1; i++) {
       if (binary_search(arr, arr+n, next_element) == 0)
       {
          res = arr[0];
          mismatch_count1++;
       }
        next_element++;
    }

    // If only one mismatch is found.
    if (mismatch_count1 == 1)
        return res;

    // If no mismatch found, elements are
    // already consecutive.
    if (mismatch_count1 == 0)
        return 0;

    // Making a series starting from last element
    // and subtracting 1 to every element.
    int mismatch_count2 = 0;
    next_element = arr[0] + n - 1;
    for (int i=n-1; i>=1; i--) {
       if (binary_search(arr, arr+n, next_element) == 0)
       {
          res = arr[n-1];
          mismatch_count2++;
```

```
       }
       next_element--;
    }

    // If only one mismatch is found.
    if (mismatch_count2 == 1)
      return res;

    return -1;
}

// Driver code
int main()
{
    int arr[] =  {7, 5, 12, 8} ;
    int n = sizeof(arr)/sizeof(arr[0]);
    int res = findElement(arr,n);
    if (res == -1)
      cout << "Answer does not exist";
    else if (res == 0)
      cout << "Elements are already consecutive";
    else
      cout << res;
    return 0;
}
```

Java

```
 // Java program to find an element
// replacement of which makes
// the array elements consecutive.
import java.io.*;
import java.util.Arrays;

class GFG
{
    static int findElement(int []arr,
                           int n)
    {
        Arrays.sort(arr);

        // Making a series starting
        // from first element and
        // adding 1 to every element.
        int mismatch_count1 = 0,
                        res = 0;
        int next_element = arr[n - 1] -
                              n + 1;
```

```
    for (int i = 0; i < n - 1; i++)
    {
    if (Arrays.binarySearch(arr,
                        next_element) < 0)
    {
        res = arr[0];
        mismatch_count1++;
    }
        next_element++;
    }

    // If only one mismatch is found.
    if (mismatch_count1 == 1)
        return res;

    // If no mismatch found, elements
    // are already consecutive.
    if (mismatch_count1 == 0)
        return 0;

    // Making a series starting
    // from last element and
    // subtracting 1 to every element.
    int mismatch_count2 = 0;
    next_element = arr[0] + n - 1;

    for (int i = n - 1; i >= 1; i--)
    {
    if (Arrays.binarySearch(arr,
                        next_element) < 0)
    {
        res = arr[n - 1];
        mismatch_count2++;
    }
    next_element--;
    }

    // If only one mismatch is found.
    if (mismatch_count2 == 1)
    return res;

    return -1;
}

// Driver code
public static void main(String args[])
{
```

```
        int []arr = new int[]{7, 5, 12, 8} ;
        int n = arr.length;
        int res = findElement(arr,n);
        if (res == -1)
        System.out.print("Answer does not exist");
        else if (res == 0)
        System.out.print("Elements are " +
                         "already consecutive");
        else
        System.out.print(res);
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
 // C# program to find an element
// replacement of which makes
// the array elements consecutive.
using System;
using System.Linq;
using System.Collections.Generic;

class GFG
{
    static int findElement(int []arr,
                           int n)
    {
        Array.Sort(arr);

        // Making a series starting
        // from first element and
        // adding 1 to every element.
        int mismatch_count1 = 0, res = 0;
        int next_element = arr[n - 1] - n + 1;

        for (int i = 0; i < n - 1; i++)
        {
        if (Array.BinarySearch(arr,
                               next_element) < 0)
        {
            res = arr[0];
            mismatch_count1++;
        }
            next_element++;
        }
```

```
    // If only one mismatch is found.
    if (mismatch_count1 == 1)
        return res;

    // If no mismatch found, elements
    // are already consecutive.
    if (mismatch_count1 == 0)
        return 0;

    // Making a series starting
    // from last element and
    // subtracting 1 to every element.
    int mismatch_count2 = 0;
    next_element = arr[0] + n - 1;

    for (int i = n - 1; i >= 1; i--)
    {
    if (Array.BinarySearch(arr,
                           next_element) < 0)
    {
        res = arr[n - 1];
        mismatch_count2++;
    }
    next_element--;
    }

    // If only one mismatch is found.
    if (mismatch_count2 == 1)
    return res;

    return -1;
}

// Driver code
static void Main()
{
    int []arr = new int[]{7, 5, 12, 8} ;
    int n = arr.Length;
    int res = findElement(arr,n);
    if (res == -1)
    Console.Write("Answer does not exist");
    else if (res == 0)
    Console.Write("Elements are " +
                  "already consecutive");
    else
    Console.Write(res);
}
```

```
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output :

```
12
```

Time Complexity : O(n Log n)

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/replacing-an-element-makes-array-elements-consecutive/

Chapter 224

Saddleback Search Algorithm in a 2D array

Saddleback Search Algorithm in a 2D array - GeeksforGeeks

Find an element in a given matrix such that each row and each column is sorted.

Examples:

```
Input : arr[] = {
                 { 1, 2, 3},
                 { 4, 5, 6},
                 { 7, 8, 9}
                }
         element=5
Output : Element Found at position (1, 1).

Input : arr[]={
              { 11, 21, 31, 41, 51 },
              { 12, 22, 32, 42, 52 },
              { 13, 23, 33, 43, 53 },
              { 14, 24, 34, 44, 54 },
              { 15, 25, 35, 45, 55 }
              }
        element=11

Output : Element Found at position (0, 0).
```

A **simple solution** is to search one by one. Time complexity of this solution is $O(n^2)$.

A **better solution** is to use Divide and Conquer to find the element. Time complexity of this solution is $O(n^{1.58})$. Please refer this article for details.

Below is an **efficient solution** that works in O(m + n) time.
1) Start with bottom left element
2) Loop: compare this element e with x
….i) if they are equal then return its position
…ii) e x then move it to right (if out of bound of matrix then break return false)
3) repeat the i), ii) and iii) till you find element or returned false

Thanks to devendraiiit for suggesting below approach.

Implementation:

C++

```
 // C++ program to search an element in row-wise
// and column-wise sorted matrix
#include<bits/stdc++.h>
using namespace std;
#define MAX 100

/* Searches the element x in mat[m][n]. If the
   element is found, then prints its position
   and returns true, otherwise prints "not found"
   and returns false */
bool search(int mat[][MAX], int m, int n, int x)
{
   int i = m-1, j = 0;  //set indexes for bottom left element
   while ( i >= 0 && j < n )
   {
      if ( mat[i][j] == x )
         return true;
      if ( mat[i][j] > x )
        i--;
      else //  if mat[i][j] < x
        j++;
   }

   return false;
}

// driver program to test above function
int main()
{
  int mat[][MAX] = { {10, 20, 30, 40},
                     {15, 25, 35, 45},
                     {27, 29, 37, 48},
                     {32, 33, 39, 50},
                     {50, 60, 70, 80},
                   };
```

```
  if (search(mat, 5, 4, 29))
      cout << "Yes";
  else
      cout << "No";
  return 0;
}
```

Java

```
 // Java program to search an
// element in row-wise and
// column-wise sorted matrix

class GFG
{
static final int MAX = 100;

/* Searches the element x
in mat[m][n]. If the element
is found, then prints its
position and returns true,
otherwise prints "not found"
and returns false */
static boolean search(int mat[][], int m,
                      int n, int x)
{

    // set indexes for
    // bottom left element
    int i = m - 1, j = 0;
        while (i >= 0 && j < n)
        {
            if (mat[i][j] == x)
                return true;
            if (mat[i][j] > x)
                i--;
            else // if mat[i][j] < x
                j++;
        }

        return false;
}

// Driver Code
public static void main(String args[])
{
int mat[][] = {{10, 20, 30, 40},
               {15, 25, 35, 45},
```

```
                {27, 29, 37, 48},
                {32, 33, 39, 50},
                {50, 60, 70, 80}};
if (search(mat, 5, 4, 29))
    System.out.println("Yes");
else
    System.out.println("No");
}
}

// This code is contributed
// by Kirti_Mangal
```

C#

```
 // C# program to search an
// element in row-wise and
// column-wise sorted matrix
using System;

class GFG
{

/* Searches the element x
in mat[m][n]. If the element
is found, then prints its
position and returns true,
otherwise prints "not found"
and returns false */
static bool search(int[,] mat, int m,
                   int n, int x)
{

    // set indexes for
    // bottom left element
    int i = m - 1, j = 0;
        while (i >= 0 && j < n)
        {
            if (mat[i, j] == x)
                return true;
            if (mat[i, j] > x)
                i--;
            else // if mat[i][j] < x
                j++;
        }

        return false;
}
```

```
// Driver Code
public static void Main()
{
int [,]mat = {{10, 20, 30, 40},
              {15, 25, 35, 45},
              {27, 29, 37, 48},
              {32, 33, 39, 50},
              {50, 60, 70, 80}};
if (search(mat, 5, 4, 29))
    Console.WriteLine("Yes");
else
    Console.WriteLine("No");
}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

Output:

```
Yes
```

Time Complexity: O(m + n)

The above can also be implemented by starting from top right corner. Please see search in a row wise and column wise sorted matrix for the alternate implementation.

Improved By : Kirti_Mangal, Abby_akku

Source

https://www.geeksforgeeks.org/saddleback-search-algorithm-in-a-2d-array/

Chapter 225

Save from Bishop in chessboard

Save from Bishop in chessboard - GeeksforGeeks

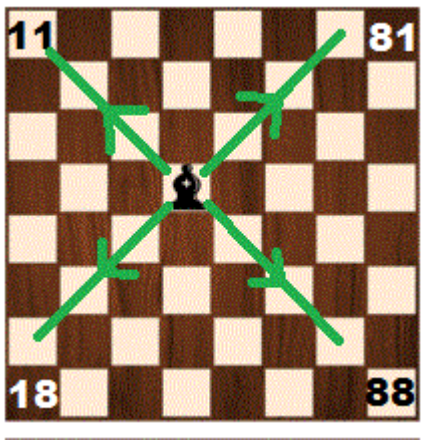
You are given a 8*8 chess board. Along with the chess board there is a Bishop placed on board and its position is known. Position of Bishop is given in form of two digit integer where both digits are greater than 0 and less than 9 (like 67 denotes 6th row and 7 column). Now your task is to find the number of ways in which you can place a pawn safely on the board.

Examples:

```
Input : Bishop's Position = 11
Output : Safe Positions = 56

Input : Bishop's Position = 44
Output : Safe Positions = 50
```

Brute Force Approach : One of the basic approach is to iterate through all the 64 possible positions on chess board and check whether that position is safe or not. This approach will require much more time.

Better Approach: As we know that movement of Bishop is in diagonal manner so from any position on the chess board a Bishop can move in both direction of both diagonals. So, all the positions which does not lie on the way of diagonal movement of the given Bishop are the safe positions.
Now our task is to find the maximum length in all possible four direction from the position of Bishop.



-> From any position a Bishop can move towards four corner that are (11, 18, 81, 88). So, we will try to find the find the maximum distance which the Bishop can move towards these corner.
Let position of bishop is ij then:

1. Distance towards 11 = min (mod(1-i), mod(1-j)).
2. Distance towards 18 = min (mod(1-i), mod(8-j)).
3. Distance towards 81 = min (mod(8-i), mod(1-j)).
4. Distance towards 88 = min (mod(8-i), mod(8-j)).

Beside all these four, one position at which Bishop is already placed is also not safe so total number of unsafe position is sum of above results + 1. and total number of safe position is 64 -(sum+1).

C++

```
 // CPP program to find total safe position
// to place your Bishop
#include<bits/stdc++.h>
using namespace std;

// function to calc total safe position
int calcSafe(int pos)
{
    // i,j denotes row and column of position of bishop
    int j = pos % 10;
    int i = pos /10;

    // calc distance in four direction
    int dis_11 = min ( abs(1-i), abs (1-j));
    int dis_18 = min ( abs(1-i), abs (8-j));
    int dis_81 = min ( abs(8-i), abs (1-j));
    int dis_88 = min ( abs(8-i), abs (8-j));
```

```
    // calc total sum of distance  + 1 for unsafe positions
    int sum = dis_11 + dis_18 + dis_81 + dis_88 + 1;

    // return total safe positions
    return (64- sum);
}

// driver function
int main()
{
    int pos = 34;
    cout << "Safe Positions = " << calcSafe(pos);
    return 0;
}
```

Java

```
 // Java program to find total safe position
// to place your Bishop
class GFG
{

    // function to calc total safe position
    static int calcSafe(int pos)
    {

        // i,j denotes row and column of position of bishop
        int j = pos % 10;
        int i = pos /10;

        // calc distance in four direction
        int dis_11 = Math.min ( Math.abs(1-i), Math.abs (1-j));
        int dis_18 = Math.min ( Math.abs(1-i), Math.abs (8-j));
        int dis_81 = Math.min ( Math.abs(8-i), Math.abs (1-j));
        int dis_88 = Math.min ( Math.abs(8-i), Math.abs (8-j));

        // calc total sum of distance + 1 for unsafe positions
        int sum = dis_11 + dis_18 + dis_81 + dis_88 + 1;

        // return total safe positions
        return (64- sum);
    }

    // Driver function
    public static void main (String[] args)
    {
        int pos = 34;
```

```
        System.out.print("Safe Positions = "+calcSafe(pos));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # python program to find total safe
# position to place your Bishop
import math

# function to calc total safe position
def calcSafe(pos):

    # i,j denotes row and column of
    # position of bishop
    j = pos % 10
    i = pos /10

    # calc distance in four direction
    dis_11 = min ( abs(1-i), abs (1-j))
    dis_18 = min ( abs(1-i), abs (8-j))
    dis_81 = min ( abs(8-i), abs (1-j))
    dis_88 = min ( abs(8-i), abs (8-j))

    # calc total sum of distance + 1
    # for unsafe positions
    sum = (dis_11 + dis_18 + dis_81
                        + dis_88 + 1)

    # return total safe positions
    return (64- sum)


# driver function
pos = 34
print("Safe Positions = " ,
            math.ceil(calcSafe(pos)))

# This code is contributed by Sam007
```

C#

```
 // Program to find the total safe
// positions to place your Bishop
using System;
```

```
class GFG {

    // function to calc total safe position
    static int calcSafe(int pos)
    {

        // i, j denotes row and column of
        // position of bishop
        int j = pos % 10;
        int i = pos / 10;

        // calc distance in four direction
        int dis_11 = Math.Min(Math.Abs(1 - i), Math.Abs(1 - j));
        int dis_18 = Math.Min(Math.Abs(1 - i), Math.Abs(8 - j));
        int dis_81 = Math.Min(Math.Abs(8 - i), Math.Abs(1 - j));
        int dis_88 = Math.Min(Math.Abs(8 - i), Math.Abs(8 - j));

        // calc total sum of distance + 1
        // for unsafe positions
        int sum = dis_11 + dis_18 + dis_81 + dis_88 + 1;

        // return total safe positions
        return (64 - sum);
    }

    // Driver function
    public static void Main()
    {
        int pos = 34;

        Console.WriteLine("Safe Positions = " + calcSafe(pos));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find
// total safe position
// to place your Bishop

// function to calculate
// total safe position
function calcSafe( $pos)
{
```

```
    // i,j denotes row and
    // column of position
    // of bishop
    $j = $pos % 10;
    $i = $pos /10;

    // calc distance in four direction
    $dis_11 = min(abs(1 - $i),
                  abs (1 - $j));
    $dis_18 = min(abs(1 - $i),
                  abs (8 - $j));
    $dis_81 = min(abs(8 - $i),
                  abs (1 - $j));
    $dis_88 = min(abs(8 - $i),
                  abs (8 - $j));

    // calc total sum of
    // distance + 1 for
    // unsafe positions
    $sum = $dis_11 + $dis_18 +
           $dis_81 + $dis_88 + 1;

    // return total safe positions
    return ceil(64- $sum);
}

    // Driver Code
    $pos = 34;
    echo "Safe Positions = " ,calcSafe($pos);

// This code is contributed by vt_m.
?>
```

Output:

```
Safe Positions = 52
```

Improved By : vt_m, Sam007

Source

https://www.geeksforgeeks.org/save-bishop-chessboard/

Chapter 226

Search an element in a sorted and rotated array

Search an element in a sorted and rotated array - GeeksforGeeks

An element in a sorted array can be found in O(log n) time via binary search. But suppose we rotate an ascending order sorted array at some pivot unknown to you beforehand. So for instance, 1 2 3 4 5 might become 3 4 5 1 2. Devise a way to find an element in the rotated array in O(log n) time.


```
Input  : arr[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};
         key = 3
Output : Found at index 8

Input  : arr[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};
         key = 30
Output : Not found

Input : arr[] = {30, 40, 50, 10, 20}
        key = 10
Output : Found at index 3
```

All solutions provided here assume that all elements in array are distinct.

The idea is to find the pivot point, divide the array in two sub-arrays and call binary search.
The main idea for finding pivot is – for a sorted (in increasing order) and pivoted array, pivot element is the only element for which next element to it is smaller than it.
Using above criteria and binary search methodology we can get pivot element in O(logn) time

```
Input arr[] = {3, 4, 5, 1, 2}
Element to Search = 1
  1) Find out pivot point and divide the array in two
      sub-arrays. (pivot = 2) /*Index of 5*/
  2) Now call binary search for one of the two sub-arrays.
      (a) If element is greater than 0th element then
             search in left array
      (b) Else Search in right array
          (1 will go in else as 1 < 0th element(3))
  3) If element is found in selected sub-array then return index
     Else return -1.
```

Below is the implementation of the above approach :

C++

```
 /* C++ Program to search an element
    in a sorted and pivoted array*/
#include <bits/stdc++.h>
using namespace std;

/* Standard Binary Search function*/
int binarySearch(int arr[], int low,
                   int high, int key)
{
  if (high < low)
    return -1;

  int mid = (low + high)/2; /*low + (high - low)/2;*/
  if (key == arr[mid])
    return mid;

  if (key > arr[mid])
    return binarySearch(arr, (mid + 1), high, key);

  // else
    return binarySearch(arr, low, (mid -1), key);
}

/* Function to get pivot. For array 3, 4, 5, 6, 1, 2
```

```
   it returns 3 (index of 6) */
int findPivot(int arr[], int low, int high)
{
  // base cases
  if (high < low) return -1;
  if (high == low) return low;

   int mid = (low + high)/2; /*low + (high - low)/2;*/
   if (mid < high && arr[mid] > arr[mid + 1])
    return mid;

   if (mid > low && arr[mid] < arr[mid - 1])
    return (mid-1);

   if (arr[low] >= arr[mid])
    return findPivot(arr, low, mid-1);

   return findPivot(arr, mid + 1, high);
}

/* Searches an element key in a pivoted
   sorted array arr[] of size n */
int pivotedBinarySearch(int arr[], int n, int key)
{
  int pivot = findPivot(arr, 0, n-1);

  // If we didn't find a pivot,
  // then array is not rotated at all
  if (pivot == -1)
    return binarySearch(arr, 0, n-1, key);

  // If we found a pivot, then first compare with pivot
  // and then search in two subarrays around pivot
  if (arr[pivot] == key)
    return pivot;

  if (arr[0] <= key)
    return binarySearch(arr, 0, pivot-1, key);

    return binarySearch(arr, pivot+1, n-1, key);
}

/* Driver program to check above functions */
int main()
{
  // Let us search 3 in below array
  int arr1[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};
  int n = sizeof(arr1)/sizeof(arr1[0]);
```

```
  int key = 3;

  // Function calling
  cout << "Index of the element is : " <<
           pivotedBinarySearch(arr1, n, key);

  return 0;
}
```

C

```
 /* Program to search an element in a sorted and pivoted array*/
#include <stdio.h>

int findPivot(int[], int, int);
int binarySearch(int[], int, int, int);

/* Searches an element key in a pivoted sorted array arrp[]
   of size n */
int pivotedBinarySearch(int arr[], int n, int key)
{
   int pivot = findPivot(arr, 0, n-1);

   // If we didn't find a pivot, then array is not rotated at all
   if (pivot == -1)
       return binarySearch(arr, 0, n-1, key);

   // If we found a pivot, then first compare with pivot and then
   // search in two subarrays around pivot
   if (arr[pivot] == key)
       return pivot;
   if (arr[0] <= key)
       return binarySearch(arr, 0, pivot-1, key);
   return binarySearch(arr, pivot+1, n-1, key);
}

/* Function to get pivot. For array 3, 4, 5, 6, 1, 2 it returns
   3 (index of 6) */
int findPivot(int arr[], int low, int high)
{
   // base cases
   if (high < low)  return -1;
   if (high == low) return low;

   int mid = (low + high)/2;   /*low + (high - low)/2;*/
   if (mid < high && arr[mid] > arr[mid + 1])
       return mid;
   if (mid > low && arr[mid] < arr[mid - 1])
```

```
        return (mid-1);
    if (arr[low] >= arr[mid])
        return findPivot(arr, low, mid-1);
    return findPivot(arr, mid + 1, high);
}

/* Standard Binary Search function*/
int binarySearch(int arr[], int low, int high, int key)
{
    if (high < low)
        return -1;
    int mid = (low + high)/2;  /*low + (high - low)/2;*/
    if (key == arr[mid])
        return mid;
    if (key > arr[mid])
        return binarySearch(arr, (mid + 1), high, key);
    return binarySearch(arr, low, (mid -1), key);
}

/* Driver program to check above functions */
int main()
{
    // Let us search 3 in below array
    int arr1[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};
    int n = sizeof(arr1)/sizeof(arr1[0]);
    int key = 3;
    printf("Index of the element is : %d",
         pivotedBinarySearch(arr1, n, key));
    return 0;
}
```

Java

```
 /* Java program to search an element
    in a sorted and pivoted array*/

class Main
{

    /* Searches an element key in a
       pivoted sorted array arrp[]
       of size n */
    static int pivotedBinarySearch(int arr[], int n, int key)
    {
       int pivot = findPivot(arr, 0, n-1);

       // If we didn't find a pivot, then
       // array is not rotated at all
```

```
    if (pivot == -1)
        return binarySearch(arr, 0, n-1, key);

    // If we found a pivot, then first
    // compare with pivot and then
    // search in two subarrays around pivot
    if (arr[pivot] == key)
        return pivot;
    if (arr[0] <= key)
        return binarySearch(arr, 0, pivot-1, key);
    return binarySearch(arr, pivot+1, n-1, key);
}

/* Function to get pivot. For array
   3, 4, 5, 6, 1, 2 it returns
   3 (index of 6) */
static int findPivot(int arr[], int low, int high)
{
    // base cases
    if (high < low)
         return -1;
    if (high == low)
         return low;

    /* low + (high - low)/2; */
    int mid = (low + high)/2;
    if (mid < high && arr[mid] > arr[mid + 1])
        return mid;
    if (mid > low && arr[mid] < arr[mid - 1])
        return (mid-1);
    if (arr[low] >= arr[mid])
        return findPivot(arr, low, mid-1);
    return findPivot(arr, mid + 1, high);
}

/* Standard Binary Search function */
static int binarySearch(int arr[], int low, int high, int key)
{
    if (high < low)
        return -1;

    /* low + (high - low)/2; */
    int mid = (low + high)/2;
    if (key == arr[mid])
        return mid;
    if (key > arr[mid])
        return binarySearch(arr, (mid + 1), high, key);
    return binarySearch(arr, low, (mid -1), key);
```

```
    }

    // main function
    public static void main(String args[])
    {
       // Let us search 3 in below array
       int arr1[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};
       int n = arr1.length;
       int key = 3;
       System.out.println("Index of the element is : "
                     + pivotedBinarySearch(arr1, n, key));
    }
}
```

Python3

```
 # Python Program to search an element
# in a sorted and pivoted array

# Searches an element key in a pivoted
# sorted array arrp[] of size n
def pivotedBinarySearch(arr, n, key):

    pivot = findPivot(arr, 0, n-1);

    # If we didn't find a pivot,
    # then array is not rotated at all
    if pivot == -1:
        return binarySearch(arr, 0, n-1, key);

    # If we found a pivot, then first
    # compare with pivot and then
    # search in two subarrays around pivot
    if arr[pivot] == key:
        return pivot
    if arr[0] <= key:
        return binarySearch(arr, 0, pivot-1, key);
    return binarySearch(arr, pivot+1, n-1, key);


# Function to get pivot. For array
# 3, 4, 5, 6, 1, 2 it returns 3
# (index of 6)
def findPivot(arr, low, high):

    # base cases
    if high < low:
        return -1
```

```
    if high == low:
        return low

    #low + (high - low)/2;
    mid = int((low + high)/2)

    if mid < high and arr[mid] > arr[mid + 1]:
        return mid
    if mid > low and arr[mid] < arr[mid - 1]:
        return (mid-1)
    if arr[low] >= arr[mid]:
        return findPivot(arr, low, mid-1)
    return findPivot(arr, mid + 1, high)

# Standard Binary Search function*/
def binarySearch(arr, low, high, key):

    if high < low:
        return -1

    #low + (high - low)/2;
    mid = int((low + high)/2)

    if key == arr[mid]:
        return mid
    if key > arr[mid]:
        return binarySearch(arr, (mid + 1), high,
                                            key);
    return binarySearch(arr, low, (mid -1), key);


# Driver program to check above functions */
# Let us search 3 in below array
arr1 = [5, 6, 7, 8, 9, 10, 1, 2, 3]
n = len(arr1)
key = 3
print("Index of the element is : ",
      pivotedBinarySearch(arr1, n, key))

# This is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# program to search an element
// in a sorted and pivoted array
using System;

class main {
```

```
// Searches an element key in a
// pivoted sorted array arrp[]
// of size n
static int pivotedBinarySearch(int[] arr,
                               int n, int key)
{
    int pivot = findPivot(arr, 0, n - 1);

    // If we didn't find a pivot, then
    // array is not rotated at all
    if (pivot == -1)
        return binarySearch(arr, 0, n - 1, key);

    // If we found a pivot, then first
    // compare with pivot and then
    // search in two subarrays around pivot
    if (arr[pivot] == key)
        return pivot;

    if (arr[0] <= key)
        return binarySearch(arr, 0, pivot - 1, key);

    return binarySearch(arr, pivot + 1, n - 1, key);
}

/* Function to get pivot. For array
3, 4, 5, 6, 1, 2 it returns
3 (index of 6) */
static int findPivot(int[] arr, int low, int high)
{
    // base cases
    if (high < low)
        return -1;
    if (high == low)
        return low;

    /* low + (high - low)/2; */
    int mid = (low + high) / 2;

    if (mid < high && arr[mid] > arr[mid + 1])
        return mid;

    if (mid > low && arr[mid] < arr[mid - 1])
        return (mid - 1);

    if (arr[low] >= arr[mid])
        return findPivot(arr, low, mid - 1);
```

```
        return findPivot(arr, mid + 1, high);
    }

    /* Standard Binary Search function */
    static int binarySearch(int[] arr, int low,
                            int high, int key)
    {
        if (high < low)
            return -1;

        /* low + (high - low)/2; */
        int mid = (low + high) / 2;

        if (key == arr[mid])
            return mid;
        if (key > arr[mid])
            return binarySearch(arr, (mid + 1), high, key);

        return binarySearch(arr, low, (mid - 1), key);
    }

    // Driver Code
    public static void Main()
    {
        // Let us search 3 in below array
        int[] arr1 = { 5, 6, 7, 8, 9, 10, 1, 2, 3 };
        int n = arr1.Length;
        int key = 3;
        Console.Write("Index of the element is : "
                    + pivotedBinarySearch(arr1, n, key));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP Program to search an element
// in a sorted and pivoted array

// Standard Binary Search function
function binarySearch($arr, $low,
                      $high, $key)
{
    if ($high < $low)
        return -1;
```

```
    /*low + (high - low)/2;*/
    $mid = floor($low + $high) / 2;

    if ($key == $arr[$mid])
        return $mid;

    if ($key > $arr[$mid])
        return binarySearch($arr, ($mid + 1),
                                $high, $key);

    else
        return binarySearch($arr, $low,
                       ($mid -1), $key);
}

// Function to get pivot.
// For array 3, 4, 5, 6, 1, 2
// it returns 3 (index of 6)
function findPivot($arr, $low, $high)
{

    // base cases
    if ($high < $low)
        return -1;
    if ($high == $low)
        return $low;

    /*low + (high - low)/2;*/
    $mid = ($low + $high)/2;
    if ($mid < $high and $arr[$mid] >
                    $arr[$mid + 1])
        return $mid;

    if ($mid > $low and $arr[$mid] <
                   $arr[$mid - 1])
        return ($mid - 1);

    if ($arr[$low] >= $arr[$mid])
        return findPivot($arr, $low,
                         $mid - 1);

    return findPivot($arr, $mid + 1, $high);
}

// Searches an element key
// in a pivoted sorted array
// arr[] of size n */
```

```
function pivotedBinarySearch($arr, $n, $key)
{
    $pivot = findPivot($arr, 0, $n - 1);

    // If we didn't find a pivot,
    // then array is not rotated
    // at all
    if ($pivot == -1)
        return binarySearch($arr, 0,
                        $n - 1, $key);

    // If we found a pivot,
    // then first compare
    // with pivot and then
    // search in two subarrays
    // around pivot
    if ($arr[$pivot] == $key)
        return $pivot;

    if ($arr[0] <= $key)
        return binarySearch($arr, 0,
                    $pivot - 1, $key);

        return binarySearch($arr, $pivot + 1,
                                $n - 1, $key);
}

// Driver Code
// Let us search 3
// in below array
$arr1 = array(5, 6, 7, 8, 9, 10, 1, 2, 3);
$n = count($arr1);
$key = 3;

// Function calling
echo "Index of the element is : " ,
      pivotedBinarySearch($arr1, $n, $key);

// This code is contributed by anuj_67.
?>
```

Output:

```
Index of the element is : 8
```

Time Complexity O(logn). Thanks to Ajay Mishra for initial solution.

Improved Solution:
We can search an element in one pass of Binary Search. The idea is to search

```
1) Find middle point mid = (l + h)/2
2) If key is present at middle point, return mid.
3) Else If arr[l..mid] is sorted
    a) If key to be searched lies in range from arr[l]
       to arr[mid], recur for arr[l..mid].
    b) Else recur for arr[mid+1..r]
4) Else (arr[mid+1..r] must be sorted)
    a) If key to be searched lies in range from arr[mid+1]
       to arr[r], recur for arr[mid+1..r].
    b) Else recur for arr[l..mid]
```

Below is the implementation of above idea :

C++

```
 // Search an element in sorted and rotated
// array using single pass of Binary Search
#include <bits/stdc++.h>
using namespace std;

// Returns index of key in arr[l..h] if
// key is present, otherwise returns -1
int search(int arr[], int l, int h, int key)
{
    if (l > h) return -1;

    int mid = (l+h)/2;
    if (arr[mid] == key) return mid;

    /* If arr[l...mid] is sorted */
    if (arr[l] <= arr[mid])
    {
        /* As this subarray is sorted, we can quickly
        check if key lies in half or other half */
        if (key >= arr[l] && key <= arr[mid])
        return search(arr, l, mid-1, key);

        return search(arr, mid+1, h, key);
    }

    /* If arr[l..mid] is not sorted, then arr[mid... r]
    must be sorted*/
    if (key >= arr[mid] && key <= arr[h])
        return search(arr, mid+1, h, key);

    return search(arr, l, mid-1, key);
```

```
}

// Driver program
int main()
{
    int arr[] = {4, 5, 6, 7, 8, 9, 1, 2, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    int key = 6;
    int i = search(arr, 0, n-1, key);

    if (i != -1)
    cout << "Index: " << i << endl;
    else
    cout << "Key not found";
}
```

Java

```
 /* Java program to search an element in
   sorted and rotated array using
   single pass of Binary Search*/

class Main
{
    // Returns index of key in arr[l..h]
    // if key is present, otherwise returns -1
    static int search(int arr[], int l, int h, int key)
    {
        if (l > h)
            return -1;

        int mid = (l+h)/2;
        if (arr[mid] == key)
            return mid;

        /* If arr[l...mid] is sorted */
        if (arr[l] <= arr[mid])
        {
            /* As this subarray is sorted, we
               can quickly check if key lies in
               half or other half */
            if (key >= arr[l] && key <= arr[mid])
               return search(arr, l, mid-1, key);

            return search(arr, mid+1, h, key);
        }

        /* If arr[l..mid] is not sorted,
```

```
           then arr[mid... r] must be sorted*/
        if (key >= arr[mid] && key <= arr[h])
            return search(arr, mid+1, h, key);

        return search(arr, l, mid-1, key);
    }

    //main function
    public static void main(String args[])
    {
        int arr[] = {4, 5, 6, 7, 8, 9, 1, 2, 3};
        int n = arr.length;
        int key = 6;
        int i = search(arr, 0, n-1, key);
        if (i != -1)
            System.out.println("Index: " + i);
        else
            System.out.println("Key not found");
    }
}
```

Python3

```
 # Search an element in sorted and rotated array using
# single pass of Binary Search

# Returns index of key in arr[l..h] if key is present,
# otherwise returns -1
def search (arr, l, h, key):
    if l > h:
        return -1

    mid = (l + h) // 2
    if arr[mid] == key:
        return mid

    # If arr[l...mid] is sorted
    if arr[l] <= arr[mid]:

        # As this subarray is sorted, we can quickly
        # check if key lies in half or other half
        if key >= arr[l] and key <= arr[mid]:
            return search(arr, l, mid-1, key)
        return search(arr, mid+1, h, key)

    # If arr[l..mid] is not sorted, then arr[mid... r]
    # must be sorted
    if key >= arr[mid] and key <= arr[h]:
```

```
        return search(a, mid+1, h, key)
    return search(arr, l, mid-1, key)

# Driver program
arr = [4, 5, 6, 7, 8, 9, 1, 2, 3]
key = 6
i = search(arr, 0, len(arr)-1, key)
if i != -1:
    print ("Index: %d"%i)
else:
    print ("Key not found")

# This code is contributed by Shreyanshi Arun
```

C#

```
 /* C# program to search an element in
sorted and rotated array using
single pass of Binary Search*/
using System;

class GFG {

    // Returns index of key in arr[l..h]
    // if key is present, otherwise
    // returns -1
    static int search(int []arr, int l, int h,
                                        int key)
    {
        if (l > h)
            return -1;

        int mid = (l + h) / 2;

        if (arr[mid] == key)
            return mid;

        /* If arr[l...mid] is sorted */
        if (arr[l] <= arr[mid])
        {

            /* As this subarray is sorted, we
            can quickly check if key lies in
            half or other half */
            if (key >= arr[l] && key <= arr[mid])
            return search(arr, l, mid - 1, key);

            return search(arr, mid + 1, h, key);
```

```
        }

        /* If arr[l..mid] is not sorted,
        then arr[mid... r] must be sorted*/
        if (key >= arr[mid] && key <= arr[h])
            return search(arr, mid + 1, h, key);

        return search(arr, l, mid - 1, key);
    }

    // main function
    public static void Main()
    {
        int []arr = {4, 5, 6, 7, 8, 9, 1, 2, 3};
        int n = arr.Length;
        int key = 6;
        int i = search(arr, 0, n - 1, key);

        if (i != -1)
            Console.WriteLine("Index: " + i);
        else
            Console.WriteLine("Key not found");
    }
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// Search an element in sorted and rotated
// array using single pass of Binary Search

// Returns index of key in arr[l..h] if
// key is present, otherwise returns -1
function search($arr, $l, $h, $key)
{
    if ($l > $h) return -1;

    $mid = ($l + $h) / 2;
    if ($arr[$mid] == $key)
        return $mid;

    /* If arr[l...mid] is sorted */
    if ($arr[$l] <= $arr[$mid])
    {

        /* As this subarray is
```

```
            sorted, we can quickly
            check if key lies in
            half or other half */
        if ($key >= $arr[$l] &&
            $key <= $arr[$mid])
                return search($arr, $l,
                       $mid - 1, $key);

        return search($arr, $mid + 1,
                           $h, $key);
    }

    /* If arr[l..mid] is not
       sorted, then arr[mid... r]
       must be sorted*/
    if ($key >= $arr[$mid] &&
          $key <= $arr[$h])
        return search($arr, $mid + 1,
                            $h, $key);

    return search($arr, $l,
             $mid-1, $key);
}

    // Driver Code
    $arr = array(4, 5, 6, 7, 8, 9, 1, 2, 3);
    $n = sizeof($arr);
    $key = 6;
    $i = search($arr, 0, $n-1, $key);

    if ($i != -1)
        echo "Index: ", floor($i) ," \n";
    else
        echo "Key not found";

// This code is contributed by ajit
?>
```

Output:

```
Index: 2
```

Thanks to Gaurav Ahirwar for suggesting above solution.

How to handle duplicates?
It doesn't look possible to search in O(Logn) time in all cases when duplicates are allowed. For example consider searching 0 in {2, 2, 2, 2, 2, 2, 2, 2, 0, 2} and {2, 0, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2}. It doesn't look possible to decide whether to recur for left half or right half by doing constant number of comparisons at the middle.

Similar Articles:
Find the minimum element in a sorted and rotated array
Given a sorted and rotated array, find if there is a pair with a given sum.

Improved By : jit_t, vt_m

Source

https://www.geeksforgeeks.org/search-an-element-in-a-sorted-and-pivoted-array/

Chapter 227

Search an element in an array where difference between adjacent elements is 1

Search an element in an array where difference between adjacent elements is 1 - GeeksforGeeks

Given an array where difference between adjacent elements is 1, write an algorithm to search for an element in the array and return the position of the element (return the first occurrence).

Examples :

```
Let element to be searched be x

Input: arr[] = {8, 7, 6, 7, 6, 5, 4, 3, 2, 3, 4, 3}
       x = 3
Output: Element 3 found at index 7

Input: arr[] =  {1, 2, 3, 4, 5, 4}
       x = 5
Output: Element 5 found at index 4
```

A **Simple Approach** is to traverse the given array one by one and compare every element with given element 'x'. If matches, then return index.

The above solution can be **Optimized** using the fact that difference between all adjacent elements is 1. The idea is to start comparing from the leftmost element and find the difference between current array element and x. Let this difference be 'diff'. From the given property of array, we always know that x must be at-least 'diff' away, so instead of searching one by one, we jump 'diff'.

Thanks to RajnishKrJha for suggesting this solution.

Below is the implementation of above idea.

C++

```
 // C++ program to search an element in an array where
// difference between all elements is 1
#include<bits/stdc++.h>
using namespace std;

// x is the elmenet to be searched in arr[0..n-1]
int search(int arr[], int n, int x)
{
    // Travers the given array starting from
    // leftmost element
    int i = 0;
    while (i<n)
    {
        // If x is found at index i
        if (arr[i] == x)
            return i;

        // Jump the difference between current
        // array element and x
        i = i + abs(arr[i]-x);
    }

    cout << "number is not present!";
    return -1;
}

// Driver program to test above function
int main()
{
    int arr[] = {8 ,7, 6, 7, 6, 5, 4, 3, 2, 3, 4, 3 };
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 3;
    cout << "Element " << x  << " is present at index "
         << search(arr,n,3);
    return 0;
}
```

Java

```
 // Java program to search an element in an
// array where difference between all
// elements is 1
```

```
import java.io.*;

class GFG {

    // x is the elmenet to be searched
    // in arr[0..n-1]
    static int search(int arr[], int n, int x)
    {

        // Travers the given array starting
        // from leftmost element
        int i = 0;
        while (i < n)
        {

            // If x is found at index i
            if (arr[i] == x)
                return i;

            // Jump the difference between current
            // array element and x
            i = i + Math.abs(arr[i]-x);
        }

        System.out.println ("number is not" +
                                    " present!");

        return -1;
    }

    // Driver program to test above function
    public static void main (String[] args) {

        int arr[] = {8 ,7, 6, 7, 6, 5, 4, 3,
                                  2, 3, 4, 3 };
        int n = arr.length;
        int x = 3;
        System.out.println("Element " + x +
                        " is present at index "
                            + search(arr,n,3));
    }
}

//This code is contributed by vt_m.
```

Python 3

```
 # Python 3 program to search an element
# in an array where difference between
# all elements is 1

# x is the elmenet to be searched in
# arr[0..n-1]
def search(arr, n, x):

    # Travers the given array starting
    # from leftmost element
    i = 0
    while (i < n):

        # If x is found at index i
        if (arr[i] == x):
            return i

        # Jump the difference between
        # current array element and x
        i = i + abs(arr[i] - x)

    print("number is not present!")
    return -1

# Driver program to test above function
arr = [8 ,7, 6, 7, 6, 5, 4, 3, 2, 3, 4, 3 ]
n = len(arr)
x = 3
print("Element" , x , " is present at index ",
                            search(arr,n,3))

# This code is contributed by Smitha
```

C#

```
 // C# program to search an element
// in an array where difference
// between all elements is 1
using System;

public class GFG
{

    // in arr[0..n-1]
    static int search(int []arr, int n,
                      int x)
    {
```

```
        // Travers the given array starting
        // from leftmost element
        int i = 0;
        while (i < n)
        {

            // If x is found at index i
            if (arr[i] == x)
                return i;

            // Jump the difference between
            // current array element and x
            i = i + Math.Abs(arr[i] - x);
        }

        Console.WriteLine ("number is not" +
                           " present!");

        return -1;
    }

    // Driver code
    public static void Main()
    {

        int []arr = {8 ,7, 6, 7, 6, 5,
                     4,3, 2, 3, 4, 3 };
        int n = arr.Length;
        int x = 3;
        Console.WriteLine("Element " + x +
                        " is present at index "
                        + search(arr, n, 3));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to search an
// element in an array where
// difference between all
// elements is 1

// x is the elmenet to be
// searched in arr[0..n-1]
function search($arr, $n, $x)
```

```
{
    // Travers the given array
    // starting from leftmost
    // element
    $i = 0;
    while ($i < $n)
    {
        // If x is found at index i
        if ($arr[$i] == $x)
            return $i;

        // Jump the difference
        // between current
        // array element and x
        $i = $i + abs($arr[$i] - $x);
    }

    echo "number is not present!";
    return -1;
}

// Driver Code
$arr = array(8 ,7, 6, 7, 6, 5,
             4, 3, 2, 3, 4, 3);
$n = sizeof($arr);
$x = 3;
echo "Element " , $x , " is present " ,
      "at index ", search($arr, $n, 3);

// This code is contributed
// by nitin mittal.
?>
```

Output :

```
Element 3 is present at index 7
```

Searching in an array where adjacent differ by at most k

This article is contributed by **Rishabh**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : Sam007, nitin mittal, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/search-an-element-in-an-array-where-difference-between-adjacent-elements-is-1/

Chapter 228

Search an element in an unsorted array using minimum number of comparisons

Search an element in an unsorted array using minimum number of comparisons - GeeksforGeeks

Given an array of **n** distinct integers and an element **x**. Search the element **x** in the array using minimum number of comparisons. Any sort of comparison will contribute 1 to the count of comparisons. For example, the condition used to terminate a loop, will also contribute 1 to the count of comparisons each time it gets executed. Expressions like **while (n) {n–;}** also contribute to the count of comparisons as value of **n** is being compared internally so as to decide whether or not to terminate the loop.

Examples:

```
Input : arr[] = {4, 6, 1, 5, 8},
        x = 1
Output : Found

Input : arr[] = {10, 3, 12, 7, 2, 11, 9},
        x = 15
Output : Not Found
```

Asked in Adobe Interview

Below simple method to search requires **2n + 1** comparisons in worst case.

```
for (i = 0; i < n; i++)  // Worst case n+1
   if (arr[i] == x)  // Worst case n
       return i;
```

How to reduce number of comparisons?
The idea is to copy x (element to be searched) to last location so that one last comparison when x is not present in arr[] is saved.

Algorithm:

```
search(arr, n, x)
    if arr[n-1] == x  // 1 comparison
       return "true"
    backup = arr[n-1]
    arr[n-1] = x

    for i=0, i++ // no termination condition
        if arr[i] == x // execute at most n times
                       // that is at-most n comparisons
            arr[n-1] = backup
            return (i < n-1) // 1 comparison
```

C/C++

```
 // C++ implementation to search an element in
// the unsorted array using minimum number of
// comparisons
#include <bits/stdc++.h>
using namespace std;

// function to search an element in
// minimum number of comparisons
string search(int arr[], int n, int x)
{
    // 1st comparison
    if (arr[n - 1] == x)
        return "Found";

    int backup = arr[n - 1];
    arr[n - 1] = x;

    // no termination condition and thus
    // no comparison
    for (int i = 0;; i++) {
        // this would be executed at-most n times
        // and therefore at-most n comparisons
        if (arr[i] == x) {
            // replace arr[n-1] with its actual element
            // as in original 'arr[]'
            arr[n - 1] = backup;
```

```
            // if 'x' is found before the '(n-1)th'
            // index, then it is present in the array
            // final comparison
            if (i < n - 1)
                return "Found";

            // else not present in the array
            return "Not Found";
        }
    }
}

// Driver program to test above
int main()
{
    int arr[] = { 4, 6, 1, 5, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 1;
    cout << search(arr, n, x);
    return 0;
}
```

Java

```
 // Java implementation to search an element in
// the unsorted array using minimum number of
// comparisons
import java.io.*;

class GFG {

    // Function to search an element in
    // minimum number of comparisons
    static String search(int arr[], int n, int x)
    {
        // 1st comparison
        if (arr[n - 1] == x)
            return "Found";

        int backup = arr[n - 1];
        arr[n - 1] = x;

        // no termination condition and thus
        // no comparison
        for (int i = 0;; i++) {
            // this would be executed at-most n times
            // and therefore at-most n comparisons
            if (arr[i] == x) {
```

```
                // replace arr[n-1] with its actual element
                // as in original 'arr[]'
                arr[n - 1] = backup;

                // if 'x' is found before the '(n-1)th'
                // index, then it is present in the array
                // final comparison
                if (i < n - 1)
                    return "Found";

                // else not present in the array
                return "Not Found";
            }
        }
    }

    // driver program
    public static void main(String[] args)
    {
        int arr[] = { 4, 6, 1, 5, 8 };
        int n = arr.length;
        int x = 1;
        System.out.println(search(arr, n, x));
    }
}

// Contributed by Pramod Kumar
```

Python3

```
 # Python3 implementation to search an
# element in the unsorted array using
# minimum number of comparisons

# function to search an element in
# minimum number of comparisons
def search(arr, n, x):

    # 1st comparison
    if (arr[n-1] == x) :
        return "Found"

    backup = arr[n-1]
    arr[n-1] = x

    # no termination condition and
    # thus no comparison
    i = 0
```

```
    while(i < n) :

        # this would be executed at-most n times
        # and therefore at-most n comparisons
        if (arr[i] == x) :

            # replace arr[n-1] with its actual
            # element as in original 'arr[]'
            arr[n-1] = backup

            # if 'x' is found before the '(n-1)th'
            # index, then it is present in the
            # array final comparison
            if (i < n-1):
                return "Found"

            # else not present in the array
            return "Not Found"
        i = i + 1

# Driver Code
arr = [4, 6, 1, 5, 8]
n = len(arr)
x = 1
print (search(arr, n, x))

# This code is contributed by rishabh_jain
```

C#

```
 // C# implementation to search an
// element in the unsorted array
// using minimum number of comparisons
using System;

class GFG {

    // Function to search an element in
    // minimum number of comparisons
    static String search(int[] arr, int n, int x)
    {
        // 1st comparison
        if (arr[n - 1] == x)
            return "Found";

        int backup = arr[n - 1];
        arr[n - 1] = x;
```

```
        // no termination condition and thus
        // no comparison
        for (int i = 0;; i++) {

            // this would be executed at-most n times
            // and therefore at-most n comparisons
            if (arr[i] == x) {

                // replace arr[n-1] with its actual element
                // as in original 'arr[]'
                arr[n - 1] = backup;

                // if 'x' is found before the '(n-1)th'
                // index, then it is present in the array
                // final comparison
                if (i < n - 1)
                    return "Found";

                // else not present in the array
                return "Not Found";
            }
        }
    }

    // driver program
    public static void Main()
    {
        int[] arr = { 4, 6, 1, 5, 8 };
        int n = arr.Length;
        int x = 1;
        Console.WriteLine(search(arr, n, x));
    }
}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP implementation to
// search an element in
// the unsorted array
// using minimum number of
// comparisons

// function to search an
// element in minimum
// number of comparisons
```

```
function search($arr, $n, $x)
{

    // 1st comparison
    if ($arr[$n - 1] == $x)
        return "Found";

    $backup = $arr[$n - 1];
    $arr[$n - 1] = $x;

    // no termination
    // condition and thus
    // no comparison
    for ($i = 0; ; $i++)
    {

        // this would be executed
        // at-most n times and
        // therefore at-most
        // n comparisons
        if ($arr[$i] == $x)
        {

            // replace arr[n-1]
            // with its actual element
            // as in original 'arr[]'
            $arr[$n - 1] = $backup;

            // if 'x' is found before
            // the '(n-1)th' index,
            // then it is present
            // in the array
            // final comparison
            if ($i < $n - 1)
                return "Found";

            // else not present
            // in the array
            return "Not Found";
        }
    }
}

// Driver Code
$arr = array( 4, 6, 1, 5, 8 );
$n = sizeof($arr);
$x = 1;
echo(search($arr, $n, $x));
```

```
// This code is contributed by Ajit.
?>
```

Output:

```
Found
```

Time Complexity: O(n)
Number of Comparisons: Atmost **(n+2)** comparisons

Improved By : Sam007, jit_t

Source

https://www.geeksforgeeks.org/search-element-unsorted-array-using-minimum-number-comparisons/

Chapter 229

Search equal, bigger or smaller in a sorted array in Java

Search equal, bigger or smaller in a sorted array in Java - GeeksforGeeks

Given array of sorted integer, search key and search preferences find array position. A search preferences can be:
1) EQUAL – search only for equal key or -1 if not found. It's a regular binary search.
2) EQUAL_OR_SMALLER – search only for equal key or the closest smaller. -1 if not found.
3) EQUAL_OR_BIGGER – search only for equal key or the closest bigger. -1 if not found.

Examples:

```
Input : { 0, 2, 4, 6 }, key -1, EQUAL
Output : -1

Input : { 0, 2, 4, 6 }, key -1, EQUAL_OR_BIGGER
Output : 0

Input : { 0, 2, 4, 6 }, key 7, EQUAL_OR_BIGGER
Output : -1

Input : { 0, 2, 4, 6 }, key 7, EQUAL_OR_SMALLER
Output : 3
```

In regular binary search algorithm evaluation and division perform as far as subarray size is bigger than 0.
In our case if we want to keep single function we need to perform final evaluation in subarray of size=2. Only in subarray size==2 is possible to check both EQUAL_OR_SMALLER and EQUAL_OR_BIGGER conditions.

In below code, **SC stands for Search Criteria**.

```
public class BinarySearchEqualOrClose {

   private static void printResult(int key, int pos,
                                   SC sC)
   {
       System.out.println("" + key + ", " + sC + ":" + pos);
   }

   enum SC {
       EQUAL,
       EQUAL_OR_BIGGER,
       EQUAL_OR_SMALLER
   };

   public static int searchEqualOrClose(int key, int[] arr, SC sC)
   {
       if (arr == null || arr.length == 0) {
           return -1;
       }

       if (arr.length == 1) { // just eliminate case of length==1

           // since algorithm needs min array size==2
           // to start final evaluations
           if (arr[0] == key) {
               return 0;
           }
           if (arr[0] < key && sC == SC.EQUAL_OR_SMALLER) {
               return 0;
           }
           if (arr[0] > key && sC == SC.EQUAL_OR_BIGGER) {
               return 0;
           }
           return -1;
       }
       return searchEqualOrClose(arr, key, 0, arr.length - 1, sC);
   }

   private static int searchEqualOrClose(int[] arr, int key,
                                         int start, int end, SC sC)
   {
       int midPos = (start + end) / 2;
       int midVal = arr[midPos];
       if (midVal == key) {
           return midPos; // equal is top priority
       }

       if (start >= end - 1) {
```

```
            if (arr[end] == key) {
                return end;
            }
            if (sC == SC.EQUAL_OR_SMALLER) {

                // find biggest of smaller element
                if (arr[start] > key && start != 0) {

                    // even before if "start" is not a first
                    return start - 1;
                }
                if (arr[end] < key) {
                    return end;
                }
                if (arr[start] < key) {
                    return start;
                }
                return -1;
            }
            if (sC == SC.EQUAL_OR_BIGGER) {

                // find smallest of bigger element
                if (arr[end] < key && end != arr.length - 1) {

                    // even after if "end" is not a last
                    return end + 1;
                }
                if (arr[start] > key) {
                    return start;
                }
                if (arr[end] > key) {
                    return end;
                }
                return -1;
            }
            return -1;
        }
        if (midVal > key) {
            return searchEqualOrClose(arr, key, start, midPos - 1, sC);
        }
        return searchEqualOrClose(arr, key, midPos + 1, end, sC);
    }

    public static void main(String[] args)
    {
        int[] arr = new int[] { 0, 2, 4, 6 };

        // test full range of xs and SearchCriteria
```

```
        for (int x = -1; x <= 7; x++) {
            int pos = searchEqualOrClose(x, arr, SC.EQUAL);
            printResult(x, pos, SC.EQUAL);
            pos = searchEqualOrClose(x, arr, SC.EQUAL_OR_SMALLER);
            printResult(x, pos, SC.EQUAL_OR_SMALLER);
            pos = searchEqualOrClose(x, arr, SC.EQUAL_OR_BIGGER);
            printResult(x, pos, SC.EQUAL_OR_BIGGER);
        }
    }
}
```

Output:

```
-1, EQUAL:-1
-1, EQUAL_OR_SMALLER:-1
-1, EQUAL_OR_BIGGER:0
0, EQUAL:0
0, EQUAL_OR_SMALLER:0
0, EQUAL_OR_BIGGER:0
1, EQUAL:-1
1, EQUAL_OR_SMALLER:0
1, EQUAL_OR_BIGGER:1
2, EQUAL:1
2, EQUAL_OR_SMALLER:1
2, EQUAL_OR_BIGGER:1
3, EQUAL:-1
3, EQUAL_OR_SMALLER:1
3, EQUAL_OR_BIGGER:2
4, EQUAL:2
4, EQUAL_OR_SMALLER:2
4, EQUAL_OR_BIGGER:2
5, EQUAL:-1
5, EQUAL_OR_SMALLER:2
5, EQUAL_OR_BIGGER:3
6, EQUAL:3
6, EQUAL_OR_SMALLER:3
6, EQUAL_OR_BIGGER:3
7, EQUAL:-1
7, EQUAL_OR_SMALLER:3
7, EQUAL_OR_BIGGER:-1
```

Time Complexity: O(log n)

Source

https://www.geeksforgeeks.org/search-equal-bigger-or-smaller-in-a-sorted-array-in-java/

Chapter 230

Search in a row wise and column wise sorted matrix

Search in a row wise and column wise sorted matrix - GeeksforGeeks

Given an n x n matrix and a number x, find the position of x in the matrix if it is present in it. Otherwise, print "Not Found". In the given matrix, every row and column is sorted in increasing order. The designed algorithm should have linear time complexity.

Example :

```
Input : mat[4][4] = { {10, 20, 30, 40},
                      {15, 25, 35, 45},
                      {27, 29, 37, 48},
                      {32, 33, 39, 50}};
              x = 29
Output : Found at (2, 1)

Input : mat[4][4] = { {10, 20, 30, 40},
                      {15, 25, 35, 45},
                      {27, 29, 37, 48},
                      {32, 33, 39, 50}};
              x = 100
Output : Element not found
```

A **simple solution** is to search one by one. Time complexity of this solution is $O(n^2)$.

A **better solution** is to use Divide and Conquer to find the element. Time complexity of this solution is $O(n^{1.58})$. Please refer this article for details.

Below is an **efficient solution** that works in O(n) time.

```
Let x = element we're trying to search for in the matrix,
    e = current element we're processing in the array.
1) Start with top right element.
2) Loop: compare this element e with x
...i) if e = x, then return position of e, since we found x in the given matrix.
...ii) if e > x then move left to check elements smaller than e (if out of bound of matrix, then
...iii) if e < x then move below to check elements greater than e (if out of bound of matrix, the
3) repeat the i), ii) and iii) until you find the element or return false
```

Thanks to devendraiiit for suggesting below approach.

Implementation:

C++

```
 // C++ program to search an element in row-wise
// and column-wise sorted matrix
#include <bits/stdc++.h>

using namespace std;

/* Searches the element x in mat[][]. If the
element is found, then prints its position
and returns true, otherwise prints "not found"
and returns false */
int search(int mat[4][4], int n, int x)
{
int i = 0, j = n-1; //set indexes for top right element
while ( i < n && j >= 0 )
{
    if ( mat[i][j] == x )
    {
        cout << "n Found at "
            << i << ", " << j;
        return 1;
    }
    if (mat[i][j] > x )
        j--;
    else // if mat[i][j] < x
        i++;
}

cout << "n Element not found";
return 0; // if ( i==n || j== -1 )
}

// Driver code
int main()
```

```
{
int mat[4][4] = { {10, 20, 30, 40},
                  {15, 25, 35, 45},
                  {27, 29, 37, 48},
                  {32, 33, 39, 50}};
search(mat, 4, 29);

return 0;
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

C

```
 // C program to search an element in row-wise
// and column-wise sorted matrix
#include<stdio.h>

/* Searches the element x in mat[][]. If the
element is found, then prints its position
and returns true, otherwise prints "not found"
and returns false */
int search(int mat[4][4], int n, int x)
{
int i = 0, j = n-1; //set indexes for top right element
while ( i < n && j >= 0 )
{
    if ( mat[i][j] == x )
    {
        printf("n Found at %d, %d", i, j);
        return 1;
    }
    if ( mat[i][j] > x )
        j--;
    else // if mat[i][j] < x
        i++;
}

printf("n Element not found");
return 0; // if ( i==n || j== -1 )
}

// driver program to test above function
int main()
{
int mat[4][4] = { {10, 20, 30, 40},
                  {15, 25, 35, 45},
```

```
                    {27, 29, 37, 48},
                    {32, 33, 39, 50},
                };
search(mat, 4, 29);
return 0;
}
```

Java

```
 // JAVA Code for Search in a row wise and
// column wise sorted matrix

class GFG{

/* Searches the element x in mat[][]. If the
element is found, then prints its position
and returns true, otherwise prints "not found"
and returns false */
    private static void search(int[][] mat, int n, int x) {

        int i = 0, j = n-1; //set indexes for top right
                            // element

        while ( i < n && j >= 0 )
        {
            if ( mat[i][j] == x )
            {
                System.out.print("n Found at "+ i + " " + j);
                return;
            }
            if ( mat[i][j] > x )
                j--;
            else // if mat[i][j] < x
                i++;
        }

        System.out.print("n Element not found");
        return; // if ( i==n || j== -1 )

    }
        // driver program to test above function
        public static void main(String[] args) {
        int mat[][] = { {10, 20, 30, 40},
                        {15, 25, 35, 45},
                        {27, 29, 37, 48},
                        {32, 33, 39, 50} };

        search(mat, 4, 29);
```

```
        }

    }
    // This code is contributed by Arnav Kr. Mandal.
```

Python3

```
 # Python3 program to search an element
# in row-wise and column-wise sorted matrix

# Searches the element x in mat[][]. If the
# element is found, then prints its position
# and returns true, otherwise prints "not found"
# and returns false
def search(mat, n, x):

    i = 0

    # set indexes for top right element
    j = n - 1
    while ( i < n and j >= 0 ):

        if (mat[i][j] == x ):

            print("n Found at ", i, ",", j)
            return 1

        if (mat[i][j] > x ):
            j -= 1

        # if mat[i][j] < x
        else:
            i += 1

    print("Element not found")
    return 0 # if (i == n || j == -1 )

# Driver Code
mat= [ [10, 20, 30, 40],
        [15, 25, 35, 45],
        [27, 29, 37, 48],
        [32, 33, 39, 50] ]
search(mat, 4, 29)

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# Code for Search in a row wise and
// column wise sorted matrix
using System;

class GFG
{
    /* Searches the element x in mat[][]. If the
    element is found, then prints its position
    and returns true, otherwise prints "not found"
    and returns false */
    private static void search(int[,] mat,
                            int n, int x)
    {
        //set indexes for top right
        // element
        int i = 0, j = n - 1;

        while ( i < n && j >= 0 )
        {
            if ( mat[i, j] == x )
            {
                Console.Write("n Found at "
                            + i + ", " + j);
                return;
            }

            if (mat[i, j] > x)
                j--;
            else // if mat[i][j] < x
                i++;
        }

        Console.Write("n Element not found");
        return; // if ( i==n || j== -1 )

    }
        // driver program to test above function
        public static void Main() {

        int [,]mat = { {10, 20, 30, 40},
                       {15, 25, 35, 45},
                       {27, 29, 37, 48},
                       {32, 33, 39, 50} };

        search(mat, 4, 29);
        }
}
```

```
// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to search an
// element in row-wise and
// column-wise sorted matrix

/* Searches the element $x
in mat[][]. If the element is
found, then prints its position
and returns true, otherwise prints
"not found" and returns false */
function search(&$mat, $n, $x)
{
    $i = 0;
    $j = $n - 1; // set indexes for
                 // top right element
    while ($i < $n && $j >= 0)
    {
        if ($mat[$i][$j] == $x)
        {
            echo "n found at " . $i.
                        ", " . $j;
            return 1;
        }
        if ($mat[$i][$j] > $x)
            $j--;
        else // if $mat[$i][$j] < $x
            $i++;
    }

    echo "n Element not found";
    return 0; // if ( $i==$n || $j== -1 )
}

// Driver Code
$mat = array(array(10, 20, 30, 40),
            array(15, 25, 35, 45),
            array(27, 29, 37, 48),
            array(32, 33, 39, 50));
search($mat, 4, 29);

// This code is contributed
// by ChitraNayal
?>
```

Output :

```
n Found at 2, 1
```

Time Complexity: O(n)

The above approach will also work for m x n matrix (not only for n x n). Complexity would be O(m + n).

Related Article :
Search element in a sorted matrix

Improved By : ChitraNayal, Abby_akku, code_master5

Source

https://www.geeksforgeeks.org/search-in-row-wise-and-column-wise-sorted-matrix/

Chapter 231

Search in an almost sorted array

Search in an almost sorted array - GeeksforGeeks

Given an array which is sorted, but after sorting some elements are moved to either of the adjacent positions, i.e., arr[i] may be present at arr[i+1] or arr[i-1]. Write an efficient function to search an element in this array. Basically the element arr[i] can only be swapped with either arr[i+1] or arr[i-1].

For example consider the array {2, 3, 10, 4, 40}, 4 is moved to next position and 10 is moved to previous position.

Example :

```
Input: arr[] =  {10, 3, 40, 20, 50, 80, 70}, key = 40
Output: 2
Output is index of 40 in given array

Input: arr[] =  {10, 3, 40, 20, 50, 80, 70}, key = 90
Output: -1
-1 is returned to indicate element is not present
```

A simple solution is to linearly search the given key in given array. Time complexity of this solution is O(n). We can modify binary search to do it in O(Logn) time.

The idea is to compare the key with middle 3 elements, if present then return the index. If not present, then compare the key with middle element to decide whether to go in left half or right half. Comparing with middle element is enough as all the elements after mid+2 must be greater than element mid and all elements before mid-2 must be smaller than mid element.

Following is the implementation of this approach.

C++

```
 // C++ program to find an element
// in an almost sorted array
#include <stdio.h>

// A recursive binary search based function.
// It returns index of x in given array
// arr[l..r] is present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
if (r >= l)
{
        int mid = l + (r - l) / 2;

        // If the element is present at
        // one of the middle 3 positions
        if (arr[mid] == x)
            return mid;
        if (mid > l && arr[mid - 1] == x)
            return (mid - 1);
        if (mid < r && arr[mid + 1] == x)
            return (mid + 1);

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 2, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid + 2, r, x);
}

// We reach here when element is not present in array
return -1;
}

// Driver Code
int main(void)
{
int arr[] = {3, 2, 10, 4, 40};
int n = sizeof(arr) / sizeof(arr[0]);
int x = 4;
int result = binarySearch(arr, 0, n - 1, x);
(result == -1) ? printf("Element is not present in array")
               : printf("Element is present at index %d",
                        result);
return 0;
}
```

Java

```
 // Java program to find an element
// in an almost sorted array
class GFG
{
    // A recursive binary search based function.
    // It returns index of x in given array
    // arr[l..r] is present, otherwise -1
    int binarySearch(int arr[], int l, int r, int x)
    {
        if (r >= l)
        {
            int mid = l + (r - l) / 2;

            // If the element is present at
            // one of the middle 3 positions
            if (arr[mid] == x)
                return mid;
            if (mid > l && arr[mid - 1] == x)
                return (mid - 1);
            if (mid < r && arr[mid + 1] == x)
                return (mid + 1);

            // If element is smaller than mid, then
            // it can only be present in left subarray
            if (arr[mid] > x)
                return binarySearch(arr, l, mid - 2, x);

            // Else the element can only be present
            // in right subarray
            return binarySearch(arr, mid + 2, r, x);
        }

        // We reach here when element is
        // not present in array
        return -1;
    }

    // Driver code
    public static void main(String args[])
    {
        GFG ob = new GFG();
        int arr[] = {3, 2, 10, 4, 40};
        int n = arr.length;
        int x = 4;
        int result = ob.binarySearch(arr, 0, n - 1, x);
        if(result == -1)
```

```
            System.out.println("Element is not present in array");
        else
            System.out.println("Element is present at index " +
                               result);
    }
}

// This code is contributed by Rajat Mishra
```

Python3

```
 # Python 3 program to find an element
# in an almost sorted array

# A recursive binary search based function.
# It returns index of x in given array arr[l..r]
# is present, otherwise -1
def binarySearch(arr, l, r, x):

    if (r >= l):

        mid = int(l + (r - l) / 2)

        # If the element is present at one
        # of the middle 3 positions
        if (arr[mid] == x): return mid
        if (mid > l and arr[mid - 1] == x):
            return (mid - 1)
        if (mid < r and arr[mid + 1] == x):
            return (mid + 1)

        # If element is smaller than mid, then
        # it can only be present in left subarray
        if (arr[mid] > x):
            return binarySearch(arr, l, mid - 2, x)

        # Else the element can only
        # be present in right subarray
        return binarySearch(arr, mid + 2, r, x)

    # We reach here when element
    # is not present in array
    return -1

# Driver Code
arr = [3, 2, 10, 4, 40]
n = len(arr)
x = 4
```

```
result = binarySearch(arr, 0, n - 1, x)
if (result == -1):
    print("Element is not present in array")
else:
    print("Element is present at index", result)

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // C# program to find an element
// in an almost sorted array
using System;

class GFG
{
    // A recursive binary search based function.
    // It returns index of x in given array
    // arr[l..r] is present, otherwise -1
    int binarySearch(int []arr, int l, int r, int x)
    {
        if (r >= l)
        {
            int mid = l + (r - l) / 2;

            // If the element is present at
            // one of the middle 3 positions
            if (arr[mid] == x)
                return mid;
            if (mid > l && arr[mid - 1] == x)
                return (mid - 1);
            if (mid < r && arr[mid + 1] == x)
                return (mid + 1);

            // If element is smaller than mid, then
            // it can only be present in left subarray
            if (arr[mid] > x)
                return binarySearch(arr, l, mid - 2, x);

            // Else the element can only be present
            // in right subarray
            return binarySearch(arr, mid + 2, r, x);
        }

        // We reach here when element is
        // not present in array
        return -1;
    }
```

```
    // Driver code
    public static void Main()
    {
        GFG ob = new GFG();
        int []arr = {3, 2, 10, 4, 40};
        int n = arr.Length;
        int x = 4;
        int result = ob.binarySearch(arr, 0, n - 1, x);
        if(result == -1)
            Console.Write("Element is not present in array");
        else
            Console.Write("Element is present at index " +
                          result);
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
 <?php
// PHP program to find an element
// in an almost sorted array

// A recursive binary search based function.
// It returns index of x in given array
// arr[l..r] is present, otherwise -1
function binarySearch($arr, $l, $r, $x)
{
    if ($r >= $l)
    {
        $mid = $l + ($r - $l) / 2;

        // If the element is present at
        // one of the middle 3 positions
        if ($arr[$mid] == $x)
            return $mid;
        if ($mid > $l && $arr[$mid - 1] == $x)
            return ($mid - 1);
        if ($mid < $r && $arr[$mid + 1] == $x)
            return ($mid + 1);

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if ($arr[$mid] > $x)
            return binarySearch($arr, $l,
                          $mid - 2, $x);
```

```
        // Else the element can only be present
        // in right subarray
        return binarySearch($arr, $mid + 2,
                                    $r, $x);
}

// We reach here when element
// is not present in array
return -1;
}

// Driver Code
$arr = array(3, 2, 10, 4, 40);
$n = sizeof($arr);
$x = 4;
$result = binarySearch($arr, 0, $n - 1, $x);
if($result == -1)
    echo("Element is not present in array");
else
    echo("Element is present at index $result");

//This code is contributed by nitin mittal
?>
```

Output :

```
Element is present at index 3
```

Time complexity of the above function is O(Logn).

This article is contributed by **Abhishek**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/search-almost-sorted-array/

Chapter 232

Search in an array of strings where non-empty strings are sorted

Search in an array of strings where non-empty strings are sorted - GeeksforGeeks

Given an array of strings. The array has both empty and non-empty strings. All non-empty strings are in sorted order. Empty strings can be present anywhere between non-empty strings.

Examples:

```
Input :  arr[] =  {"for", "", "", "", "geeks",
                   "ide", "", "practice", "" ,
                   "", "quiz", "", ""};
          str = "quiz"
Output :   10
The string "quiz" is present at index 10 in
given array.
```

A **simple solution** is to linearly search given str in array of strings.

A **better solution** is to do modified Binary Search. Like normal binary search, we compare given str with middle string. If middle string is empty, we find the closest non-empty string x (by linearly searching on both sides). Once we find x, we do standard binary search, i.e., we compare given str with x. If str is same as x, we return index of x. if str is greater, we recur for right half, else we recur for left half.

Below is C++ implementation of the idea.

```
// C++ program to find location of a str in
```

```
// an array of strings which is sorted and
// has empty strings between strings.
#include <bits/stdc++.h>
using namespace std;

// Compare two string equals are not
int compareStrings(string str1, string str2)
{
    int i = 0;
    while (str1[i] == str2[i] && str1[i] != '\0')
        i++;
    if (str1[i] > str2[i])
        return -1;

    return (str1[i] < str2[i]);
}

// Main function to find string location
int searchStr(string arr[], string str, int first,
                                        int last)
{
    if (first > last)
        return -1;

    // Move mid to the middle
    int mid = (last+first)/2;

    // If mid is empty , find closet non-empty string
    if (arr[mid].empty())
    {
        // If mid is empty, search in both sides of mid
        // and find the closest non-empty string, and
        // set mid accordingly.
        int left  = mid - 1;
        int right = mid + 1;
        while (true)
        {
            if (left < first && right > last)
                return -1;
            if (right<=last && !arr[right].empty())
            {
                mid = right;
                break;
            }
            if (left>=first && !arr[left].empty())
            {
                mid = left;
                break;
```

```
            }
            right++;
            left--;
        }
    }

    // If str is found at mid
    if (compareStrings(str, arr[mid]) == 0)
        return mid;

    // If str is greater than mid
    if (compareStrings(str, arr[mid]) < 0)
        return searchStr(arr, str, mid+1, last);

    // If str is smaller than mid
    return searchStr(arr, str, first, mid-1);
}

// Driver Code
int main()
{
    // Input arr of Strings.
    string arr[] = {"for", "", "", "", "geeks", "ide", "",
                    "practice", "" , "", "quiz", "", ""};

    // input Search String
    string str = "quiz";
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << searchStr(arr, str, 0, n-1);
    return 0;
}
```

Output:

```
  10
```

Although this approach works better than linear search, the worst-case runtime for this algorithm is O(n).

Source

https://www.geeksforgeeks.org/search-in-an-array-of-strings-where-non-empty-strings-are-sorted/

Chapter 233

Search, insert and delete in a sorted array

Search, insert and delete in a sorted array - GeeksforGeeks

In this post search, insert and delete operation in a sorted array is discussed.

Search Operation

In a sorted array, the search operation can be performed by using binary search.

C

```
 // C program to implement binary search in sorted array
#include<stdio.h>

int binarySearch(int arr[], int low, int high, int key)
{
   if (high < low)
       return -1;
   int mid = (low + high)/2;  /*low + (high - low)/2;*/
   if (key == arr[mid])
       return mid;
   if (key > arr[mid])
       return binarySearch(arr, (mid + 1), high, key);
   return binarySearch(arr, low, (mid -1), key);
}

/* Driver program to check above functions */
int main()
{
   // Let us search 3 in below array
   int arr[] = {5, 6, 7, 8, 9, 10};
```

```
    int n,key;

   n = sizeof(arr)/sizeof(arr[0]);
   key =10;

   printf("Index: %d\n", binarySearch(arr,0, n, key) );
   return 0;
}
```

Java

```
 // Java program to implement binary
// search in a sorted array

class Main
{
    // function to implement
    // binary search
    static int binarySearch(int arr[], int low, int high, int key)
    {
       if (high < low)
           return -1;

        /*low + (high - low)/2;*/
       int mid = (low + high)/2;
       if (key == arr[mid])
           return mid;
       if (key > arr[mid])
           return binarySearch(arr, (mid + 1), high, key);
       return binarySearch(arr, low, (mid -1), key);
    }

    /* Driver program to test above function */
    public static void main (String[] args)
    {
       int arr[] = {5, 6, 7, 8, 9, 10};
       int n,key;
       n = arr.length;
       key =10;

       System.out.println("Index: " +
           binarySearch(arr,0, n, key) );
    }
}
```

Python3

```
 # python 3  program to implement
```

```
# binary search in sorted array

def binarySearch(arr, low, high, key):

    if (high < low):
        return -1
    # low + (high - low)/2
    mid = (low + high)/2

    if (key == arr[int(mid)]):
        return mid

    if (key > arr[int(mid)]):
        return binarySearch(arr,
           (mid + 1), high, key)

    return (binarySearch(arr, low,
           (mid -1), key))

# Driver program to check above functions
# Let us search 3 in below array
arr = [5, 6, 7, 8, 9, 10]
n = len(arr)
key =10
print("Index:", int(binarySearch(arr,0, n, key) ))

# This code is contributed by
# Smitha Dinesh Semwal
```

Output:

```
Index: 5
```

Insert Operation

In an unsorted array, the insert operation is faster as compared to sorted array because we don't have to care about the position at which the element to be placed.

C

```
 // C program to implement insert operation in
// an sorted array.
#include<stdio.h>

// Inserts a key in arr[] of given capacity.  n is current
// size of arr[]. This function returns n+1 if insertion
```

```
// is successful, else n.
int insertSorted(int arr[], int n, int key, int capacity)
{
    // Cannot insert more elements if n is already
    // more than or equal to capcity
    if (n >= capacity)
       return n;

    int i;
    for (i=n-1; ( i >= 0  && arr[i] > key); i--)
       arr[i+1] = arr[i];

    arr[i+1] = key;

    return (n+1);
}

/* Driver program to test above function */
int main()
{
    int arr[20] = {12, 16, 20, 40, 50, 70};
    int capacity = sizeof(arr)/sizeof(arr[0]);
    int n = 6;
    int i, key = 26;

    printf("\nBefore Insertion: ");
    for (i=0; i<n; i++)
        printf("%d  ", arr[i]);

    // Inserting key
    n = insertSorted(arr, n, key, capacity);

    printf("\nAfter Insertion: ");
    for (i=0; i<n; i++)
        printf("%d  ",arr[i]);

    return 0;
}
```

Java

```
 // Java program to insert an
// element in a sorted array

class Main
{
    // Inserts a key in arr[] of given
    // capacity.  n is current size of arr[].
```

```
    // This function returns n+1 if insertion
    // is successful, else n.
    static int insertSorted(int arr[], int n, int key, int capacity)
    {
        // Cannot insert more elements if n is already
        // more than or equal to capcity
        if (n >= capacity)
           return n;

        int i;
        for (i=n-1; (i >= 0 && arr[i] > key); i--)
           arr[i+1] = arr[i];

        arr[i+1] = key;

        return (n+1);
    }

    /* Driver program to test above function */
    public static void main (String[] args)
    {
        int arr[] = new int[20];
        arr[0] = 12;
        arr[1] = 16;
        arr[2] = 20;
        arr[3] = 40;
        arr[4] = 50;
        arr[5] = 70;
        int capacity = arr.length;
        int n = 6;
        int key = 26;

    System.out.print("\nBefore Insertion: ");
    for (int i=0; i<n; i++)
        System.out.print(arr[i] + " ");

    // Inserting key
    n = insertSorted(arr, n, key, capacity);

    System.out.print("\nAfter Insertion: ");
    for (int i=0; i<n; i++)
        System.out.print(arr[i] + " ");
    }
}
```

Output:

```
Before Insertion: 12 16 20 40 50 70
After Insertion: 12 16 20 26 40 50 70
```

Delete Operation

In delete operation, the element to be deleted is searched using binary search and then delete operation is performed followed by shifting the elements.

C

```
 // C program to implement delete operation in a
// sorted array
#include<stdio.h>

// To search a ley to be deleted
int binarySearch(int arr[], int low, int high, int key);

/* Function to delete an element */
int deleteElement(int arr[], int n, int key)
{
    // Find position of element to be deleted
    int pos = binarySearch(arr, 0, n-1, key);

    if (pos==-1)
    {
        printf("Element not found");
        return n;
    }

    // Deleting element
    int i;
    for (i=pos; i<n; i++)
        arr[i] = arr[i+1];

    return n-1;
}

int binarySearch(int arr[], int low, int high, int key)
{
    if (high < low)
        return -1;
    int mid = (low + high)/2;
    if (key == arr[mid])
        return mid;
    if (key > arr[mid])
        return binarySearch(arr, (mid + 1), high, key);
    return binarySearch(arr, low, (mid -1), key);
}
```

```
// Driver code
int main()
{
    int i;
    int arr[] = {10, 20, 30, 40, 50};

    int n = sizeof(arr)/sizeof(arr[0]);
    int key = 30;

    printf("Array before deletion\n");
    for (i=0; i<n; i++)
      printf("%d  ", arr[i]);

    n = deleteElement(arr, n, key);

    printf("\n\nArray after deletion\n");
    for (i=0; i<n; i++)
      printf("%d  ", arr[i]);
}
```

Java

```
 // Java program to delete an
// element from a sorted array

class Main
{
    // binary search
    static int binarySearch(int arr[], int low, int high, int key)
    {
        if (high < low)
            return -1;
        int mid = (low + high)/2;
        if (key == arr[mid])
            return mid;
        if (key > arr[mid])
            return binarySearch(arr, (mid + 1), high, key);
        return binarySearch(arr, low, (mid -1), key);
    }

    /* Function to delete an element */
    static int deleteElement(int arr[], int n, int key)
    {
        // Find position of element to be deleted
        int pos = binarySearch(arr, 0, n-1, key);

        if (pos==-1)
        {
```

```
            System.out.println("Element not found");
            return n;
        }

        // Deleting element
        int i;
        for (i=pos; i<n-1; i++)
            arr[i] = arr[i+1];

        return n-1;
    }

    /* Driver program to test above function */
    public static void main (String[] args)
    {

        int i;
        int arr[] = {10, 20, 30, 40, 50};

        int n = arr.length;
        int key = 30;

        System.out.print("Array before deletion:\n");
        for (i=0; i<n; i++)
          System.out.print(arr[i] + " ");

        n = deleteElement(arr, n, key);

        System.out.print("\n\nArray after deletion:\n");
        for (i=0; i<n; i++)
          System.out.print(arr[i] + " ");
    }
}
```

Output:

```
Array before deletion
10 20 30 40 50

Array after deletion
10 20 40 50
```

Search: O(Log n) [Using Binary Search]
Insert: O(n) [In worst case all elements may have to be moved]
Delete: O(n) [In worst case all elements may have to be moved]

Source

https://www.geeksforgeeks.org/search-insert-and-delete-in-a-sorted-array/

Chapter 234

Search, insert and delete in an unsorted array

Search, insert and delete in an unsorted array - GeeksforGeeks

In this post search, insert and delete operation in an unsorted array is discussed.

Search Operation

In an unsorted array, the search operation can be performed by linear traversal from the first element to the last element.

C

```
 // C program to implement linear
// search in unsorted array
#include<stdio.h>

// Function to implement search operation
int findElement(int arr[], int n,
                int key)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == key)
            return i;

    return -1;
}

// Driver Code
int main()
{
    int arr[] = {12, 34, 10, 6, 40};
```

```
    int n = sizeof(arr) / sizeof(arr[0]);

    // Using a last element as search element
    int key = 40;
    int position = findElement(arr, n, key);

    if (position == - 1)
        printf("Element not found");
    else
        printf("Element Found at Position: %d", position + 1 );

    return 0;
}
```

Java

```
 // Java program to implement linear
// search in unsorted arrays

class Main
{
    // Function to implement
    // search operation
    static int findElement(int arr[], int n,
                           int key)
    {
        for (int i = 0; i < n; i++)
            if (arr[i] == key)
                return i;

        return -1;
    }

    // Driver Code
    public static void main(String args[])
    {
        int arr[] = {12, 34, 10, 6, 40};
        int n = arr.length;

        // Using a last element as search element
        int key = 40;
        int position = findElement(arr, n, key);

        if (position == - 1)
            System.out.println("Element not found");
        else
            System.out.println("Element Found at Position: "
                               + (position + 1));
```

```
    }
}
```

Python

```
 # Python program for searching in
# unsorted array

def findElement(arr, n, key):
    for i in range (n):
        if (arr[i] == key):
            return i
    return -1

arr = [12, 34, 10, 6, 40]
key = 40
n = len(arr)

#search operation
index = findElement(arr, n, key)
if index != -1:
    print ("element found at position: " + str(index + 1 ))
else:
    print ("element not found")

# Thanks to Aditi Sharma for contributing
# this code
```

C#

```
 // C# program to implement linear
// search in unsorted arrays
using System;

class main
{
    // Function to implement
    // search operation
    static int findElement(int []arr, int n,
                           int key)
    {
        for (int i = 0; i < n; i++)
            if (arr[i] == key)
                return i;

        return -1;
    }
```

```
    // Driver Code
    public static void Main()
    {
        int []arr = {12, 34, 10, 6, 40};
        int n = arr.Length;

        // Using a last element as
        // search element
        int key =40;
        int position = findElement(arr,n,key);

        if (position == - 1)
        Console.WriteLine("Element not found");
        else
        Console.WriteLine("Element Found at Position: "
                           + (position+1));
    }
}

//  This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to implement linear
// search in unsorted array

// Function to implement
// search operation
function findElement($arr, $n, $key)
{
    $i;
    for ($i = 0; $i < $n; $i++)
        if ($arr[$i] == $key)
            return $i;

    return -1;
}

// Driver Code
$arr = array(12, 34, 10, 6, 40);
$n = sizeof($arr);

// Using a last element
// as search element
$key = 40;
$position = findElement($arr, $n, $key);
```

```
if ($position == - 1)
    echo("Element not found");
else
    echo("Element Found at Position: " . ($position + 1));

// This code is contributed by Ajit.
?>
```

Output:

```
Element Found at Position: 5
```

Insert Operation

In an unsorted array, the insert operation is faster as compared to sorted array because we don't have to care about the position at which the element is to be placed.

C

```
 // C program to implement insert
// operation in an unsorted array.
#include<stdio.h>

// Inserts a key in arr[] of given capacity.
// n is current size of arr[]. This
// function returns n + 1 if insertion
// is successful, else n.
int insertSorted(int arr[], int n,
                 int key,
                 int capacity)
{

    // Cannot insert more elements if n is
    // already more than or equal to capcity
    if (n >= capacity)
       return n;

    arr[n] = key;

    return (n + 1);
}

// Driver Code
int main()
{
```

```
    int arr[20] = {12, 16, 20, 40, 50, 70};
    int capacity = sizeof(arr) / sizeof(arr[0]);
    int n = 6;
    int i, key = 26;

    printf("\n Before Insertion: ");
    for (i = 0; i < n; i++)
        printf("%d  ", arr[i]);

    // Inserting key
    n = insertSorted(arr, n, key, capacity);

    printf("\n After Insertion: ");
    for (i = 0; i < n; i++)
        printf("%d  ",arr[i]);

    return 0;
}
```

Java

```
 // Java program to implement insert
// operation in an unsorted array.

class Main
{
    // Function to insert a given key in
    // the array. This function returns n+1
    // if insertion is successful, else n.
    static int insertSorted(int arr[], int n,
                            int key,
                            int capacity)
    {

        // Cannot insert more elements if n
        // is already more than or equal to
        // capcity
        if (n >= capacity)
           return n;

        arr[n] = key;

        return (n + 1);
    }

    // Driver Code
    public static void main (String[] args)
    {
```

```
        int[] arr = new int[20];
        arr[0] = 12;
        arr[1] = 16;
        arr[2] = 20;
        arr[3] = 40;
        arr[4] = 50;
        arr[5] = 70;
        int capacity = 20;
        int n = 6;
        int i, key = 26;

        System.out.print("Before Insertion: ");
        for (i = 0; i < n; i++)
            System.out.print(arr[i]+" ");

        // Inserting key
        n = insertSorted(arr, n, key, capacity);

        System.out.print("\n After Insertion: ");
        for (i = 0; i < n; i++)
            System.out.print(arr[i]+" ");
    }
}
```

Python

```
 # Python program for inserting
# an element in an unsorted array

# method to insert element
def insert(arr, element):
    arr.append(element)

# declaring array and key to insert
arr = [12, 16, 20, 40, 50, 70]
key = 26

# array before inserting an element
print ("Before Inserting: ")
print (arr)

# array after Inserting element
insert(arr, key)
print("After Inserting: ")
print (arr)

# Thanks to Aditi Sharma for contributing
# this code
```

C#

```
 // C# program to implement insert
// operation in an unsorted array.
using System;

class main
{

    // Function to insert a given
    // key in the array. This
    // function  returns n + 1
    // if insertion is successful,
    // else n.
    static int insertSorted(int []arr, int n,
                            int key,
                            int capacity)
    {

        // Cannot insert more elements
        // if n is already more than
        // or equal to capcity
        if (n >= capacity)
        return n;

        arr[n] = key;
        return (n + 1);
    }

    // Driver Code
    public static void Main ()
    {
        int[] arr = new int[20];
        arr[0] = 12;
        arr[1] = 16;
        arr[2] = 20;
        arr[3] = 40;
        arr[4] = 50;
        arr[5] = 70;
        int capacity = 20;
        int n = 6;
        int i, key = 26;

        Console.Write("Before Insertion: ");
        for (i = 0; i < n; i++)
            Console.Write(arr[i]+" ");
            Console.WriteLine();
```

```
        // Inserting key
        n = insertSorted(arr, n, key, capacity);

        Console.Write("After Insertion: ");
        for (i = 0; i < n; i++)
            Console.Write(arr[i]+" ");

    }
}

// This code is contributed by vt_m.
```

Output:

```
Before Insertion: 12 16 20 40 50 70
After Insertion: 12 16 20 40 50 70 26
```

Delete Operation

In delete operation, the element to be deleted is searched using the linear search and then delete operation is performed followed by shifting the elements.

C

```
 // C program to implement delete operation in a
// unsorted array
#include<stdio.h>

// To search a key to be deleted
int findElement(int arr[], int n,
                int key);

// Function to delete an element
int deleteElement(int arr[], int n,
                  int key)
{
    // Find position of element to be deleted
    int pos = findElement(arr, n, key);

    if (pos == - 1)
    {
        printf("Element not found");
        return n;
    }

    // Deleting element
```

```
    int i;
    for (i = pos; i < n - 1; i++)
        arr[i] = arr[i + 1];

    return n - 1;
}

// Function to implement search operation
int findElement(int arr[], int n, int key)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == key)
            return i;

    return - 1;
}

// Driver code
int main()
{
    int i;
    int arr[] = {10, 50, 30, 40, 20};

    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 30;

    printf("Array before deletion\n");
    for (i = 0; i < n; i++)
      printf("%d  ", arr[i]);

    n = deleteElement(arr, n, key);

    printf("\nArray after deletion\n");
    for (i = 0; i < n; i++)
      printf("%d  ", arr[i]);

    return 0;
}
```

Java

```
 // Java program to implement delete
// operation in an unsorted array

class Main
{
    // function to search a key to
```

```
// be deleted
static int findElement(int arr[], int n, int key)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == key)
            return i;

    return -1;
}

// Function to delete an element
static int deleteElement(int arr[], int n, int key)
{
    // Find position of element to be
    // deleted
    int pos = findElement(arr, n, key);

    if (pos == -1)
    {
        System.out.println("Element not found");
        return n;
    }

    // Deleting element
    int i;
    for (i = pos; i< n - 1; i++)
        arr[i] = arr[i + 1];

    return n - 1;
}

// Driver Code
public static void main(String args[])
{
    int i;
    int arr[] = {10, 50, 30, 40, 20};

    int n = arr.length;
    int key = 30;

    System.out.println("Array before deletion");
    for (i=0; i<n; i++)
      System.out.print(arr[i] + " ");

    n = deleteElement(arr, n, key);

    System.out.println("\n\nArray after deletion");
```

```
        for (i=0; i<n; i++)
          System.out.print(arr[i]+" ");
    }
}
```

Python

```
 # Python program to delete an element
# from an unsorted array

# Declaring array and key to delete
arr = [10, 50, 30, 40, 20]
key = 30

print("Array before dletion:")
print arr

# deletes key if found in the array
# otherwise shows error not in list
arr.remove(key)
print("Array after deletion")
print(arr)

# This code is contributed by Aditi Sharma.
```

C#

```
 // C# program to implement delete
// operation in an unsorted array
using System;

class main
{
    // Function to search a
    // key to be deleted
    static int findElement(int []arr,
                           int n,
                           int key)
    {

        int i;
        for (i = 0; i < n; i++)
            if (arr[i] == key)
                return i;

        return -1;
    }
```

```
    // Function to delete an element
    static int deleteElement(int []arr,
                             int n,
                             int key)
    {
        // Find position of element
        // to be deleted
        int pos = findElement(arr, n, key);

        if (pos == - 1)
        {
            Console.WriteLine("Element not found");
            return n;
        }

        // Deleting element
        int i;
        for (i = pos; i < n - 1; i++)
            arr[i] = arr[i + 1];

        return n - 1;
    }

    // Driver Code
    public static void Main()
    {
        int i;
        int []arr = {10, 50, 30, 40, 20};

        int n = arr.Length;
        int key = 30;

        Console.Write("Array before deletion ");
        for (i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
        Console.WriteLine();

        n = deleteElement(arr, n, key);

        Console.Write("Array after deletion ");
        for (i = 0; i < n; i++)
        Console.Write(arr[i]+" ");
    }
}

// This code is contributed by vt_m.
```

Output:

```
Array before deletion
10 50 30 40 20

Array after deletion
10 50 40 20
```

Time complexities:
Search: O(n)
Insert: O(1)
Delete: O(n)

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/search-insert-and-delete-in-an-unsorted-array/

Chapter 235

Searching in an array where adjacent differ by at most k

Searching in an array where adjacent differ by at most k - GeeksforGeeks

A step array is an array of integer where each element has a difference of atmost k with its neighbor. Given a key x, we need to find the index value of k if multiple element exist return the first occurrence of key.

Examples:

```
Input : arr[] = {4, 5, 6, 7, 6}
           k = 1
           x = 6
Output : 2
The first index of 6 is 2.

Input : arr[] = {20, 40, 50, 70, 70, 60}
          k = 20
          x = 60
Output : 5
The index of 60 is 5
```

This problem is mainly an extension of Search an element in an array where difference between adjacent elements is 1.

A **Simple Approach** is to traverse the given array one by one and compare every element with given element 'x'. If matches, then return index.

The above solution can be **Optimized** using the fact that difference between all adjacent elements is at most k. The idea is to start comparing from the leftmost element and find the difference between current array element and x. Let this difference be 'diff'. From the

given property of array, we always know that x must be at-least 'diff/k' away, so instead of searching one by one, we jump 'diff/k'.

Below is the implementation of above idea.

C++

```
 // C++ program to search an element in an array
// where difference between all elements is 1
#include<bits/stdc++.h>
using namespace std;

// x is the element to be searched in arr[0..n-1]
// such that all elements differ by at-most k.
int search(int arr[], int n, int x, int k)
{
    // Travers the given array starting from
    // leftmost element
    int i = 0;
    while (i < n)
    {
        // If x is found at index i
        if (arr[i] == x)
            return i;

        // Jump the difference between current
        // array element and x divided by k
        // We use max here to make sure that i
        // moves at-least one step ahead.
        i = i + max(1, abs(arr[i]-x)/k);
    }

    cout << "number is not present!";
    return -1;
}

// Driver program to test above function
int main()
{
    int arr[] = {2, 4, 5, 7, 7, 6};
    int x = 6;
    int k = 2;
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Element " << x  << " is present at index "
         << search(arr, n, x, k);
    return 0;
}
```

Java

```
 // Java program to search an element in
// an array where difference between all
// elements is 1

import java.io.*;

class GFG {

    // x is the element to be searched
    // in arr[0..n-1] such that all
    // elements differ by at-most k.
    static int search(int arr[], int n,
                            int x, int k)
    {

        // Travers the given array starting
        // from leftmost element
        int i = 0;

        while (i < n) {

            // If x is found at index i
            if (arr[i] == x)
                return i;

            // Jump the difference between
            // current array element and x
            // divided by k We use max here
            // to make sure that i moves
            // at-least one step ahead.
            i = i + Math.max(1, Math.abs(arr[i]
                                    - x) / k);
        }

        System.out.println("number is " +
                                "not present!");
        return -1;
    }

    // Driver program to test above function
    public static void main(String[] args)
    {

        int arr[] = { 2, 4, 5, 7, 7, 6 };
        int x = 6;
        int k = 2;
        int n = arr.length;
```

```
        System.out.println("Element " + x +
                           " is present at index "
                           + search(arr, n, x, k));
    }
}

// This code is contributed by vt_m
```

Python3

```
 # Python 3 program to search an element in an array
# where difference between all elements is 1

# x is the element to be searched in arr[0..n-1]
# such that all elements differ by at-most k.
def search(arr, n, x, k):

    # Travers the given array starting from
    # leftmost element
    i = 0
    while (i < n):

        # If x is found at index i
        if (arr[i] == x):
            return i

        # Jump the difference between current
        # array element and x divided by k
        # We use max here to make sure that i
        # moves at-least one step ahead.
        i = i + max(1, int(abs(arr[i] - x) / k))


    print("number is not present!")
    return -1


# Driver program to test above function
arr = [2, 4, 5, 7, 7, 6]
x = 6
k = 2
n = len(arr)
print("Element", x, "is present at index",search(arr, n, x, k))

# This code is contributed
# by Smitha Dinesh Semwal
```

C#

```
 // C# program to search an element in
// an array where difference between
// all elements is 1
using System;

class GFG {

    // x is the element to be searched
    // in arr[0..n-1] such that all
    // elements differ by at-most k.
    static int search(int []arr, int n,
                          int x, int k)
    {

        // Travers the given array starting
        // from leftmost element
        int i = 0;

        while (i < n)
        {

            // If x is found at index i
            if (arr[i] == x)
                return i;

            // Jump the difference between
            // current array element and x
            // divided by k We use max here
            // to make sure that i moves
            // at-least one step ahead.
            i = i + Math.Max(1, Math.Abs(arr[i]
                                    - x) / k);
        }

        Console.Write("number is " +
                      "not present!");
        return -1;
    }

    // Driver Code
    public static void Main()
    {

        int []arr = { 2, 4, 5, 7, 7, 6 };
        int x = 6;
        int k = 2;
        int n = arr.Length;
```

```
        Console.Write("Element " + x +
                      " is present at index " +
                        search(arr, n, x, k));
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
 <?php
// PHP program to search an
// element in an array where
// difference between all
// elements is 1

// x is the element to be
// searched in arr[0..n-1]
// such that all elements
// differ by at-most k.
function search($arr, $n, $x, $k)
{

    // Travers the given array
    // starting from leftmost element
    $i = 0;
    while ($i < $n)
    {
        // If x is found at index i
        if ($arr[$i] == $x)
            return $i;

        // Jump the difference between current
        // array element and x divided by k
        // We use max here to make sure that i
        // moves at-least one step ahead.
        $i = $i + max(1, abs($arr[$i] - $x) / $k);
    }

    echo "number is not present!";
    return -1;
}

// Driver Code
{
    $arr = array(2, 4, 5, 7, 7, 6);
    $x = 6;
    $k = 2;
```

```
    $n = sizeof($arr)/sizeof($arr[0]);
    echo "Element $x is present".
                    "at index ",
        search($arr, $n, $x, $k);
    return 0;
}

// This code is contributed by nitin mittal.
?>
```

Output:

```
Element 6 is present at index 5
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/searching-array-adjacent-differ-k/

Chapter 236

Second minimum element using minimum comparisons

Second minimum element using minimum comparisons - GeeksforGeeks

Given an array of integers, find the minimum (or maximum) element and the element just greater (or smaller) than that in less than 2n comparisons. The given array is not necessarily sorted. Extra space is allowed.

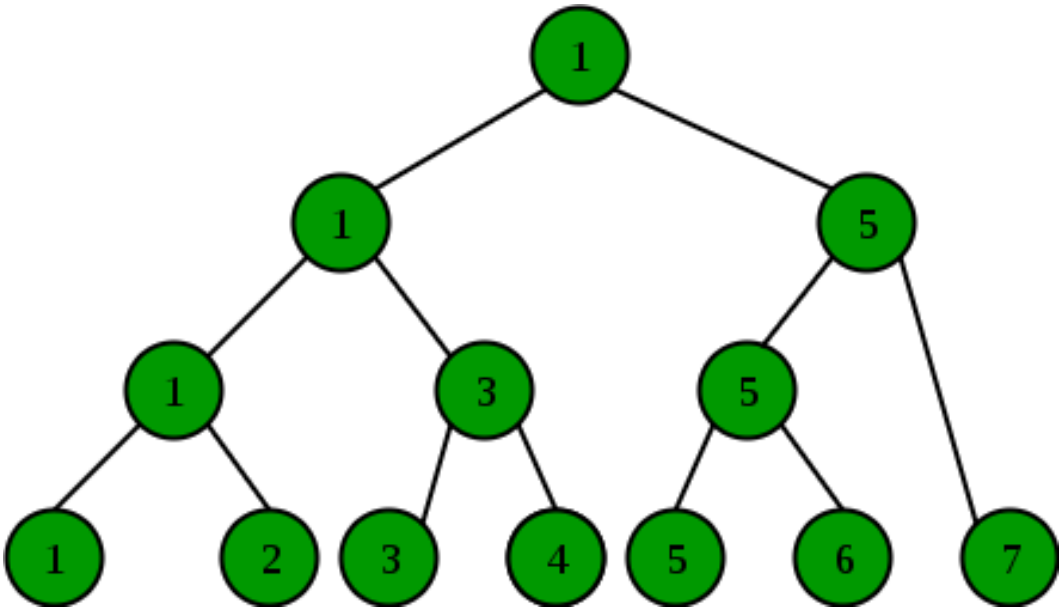
Examples:

```
Input: {3, 6, 100, 9, 10, 12, 7, -1, 10}
Output: Minimum: -1, Second minimum: 3
```

We have already discussed an approach in below post.
Find the smallest and second smallest element in an array

Comparisons of array elements can be costly if array elements are of large size, for example large strings. We can minimize the number of comparisons used in above approach.

The idea is based on tournament tree. Consider each element in the given array as a leaf node.



First, we find the minimum element as explained by building a tournament tee. To build the tree, we compare all adjacent pairs of elements (leaf nodes) with each other. Now we have n/2 elements which are smaller than their counterparts (from each pair, the smaller element forms the level above that of the leaves). Again, find the smaller elements in each of the n/4 pairs. Continue this process until the root of the tree is created. The root is the minimum.
Next, we traverse the tree and while doing so, we discard the sub trees whose root is greater than the smallest element. Before discarding, we update the second smallest element, which will hold our result. The point to understand here is that the tree is height balanced and we have to spend only logn time to traverse all the elements that were ever compared to the minimum in step 1. Thus, the overall time complexity is **n + logn**. Auxiliary space needed is O(2n), as the number of leaf nodes will be approximately equal to the number of internal nodes.

```
 // C++ program to find minimum and second minimum
// using minimum number of comparisons
#include <bits/stdc++.h>
using namespace std;

// Tournament Tree node
struct Node
{
    int idx;
    Node *left, *right;
};

// Utility function to create a tournament tree node
Node *createNode(int idx)
```

```
{
    Node *t = new Node;
    t->left = t->right = NULL;
    t->idx = idx;
    return t;
}

// This function traverses tree across height to
// find second smallest element in tournament tree.
// Note that root is smallest element of tournament
// tree.
void traverseHeight(Node *root, int arr[], int &res)
{
    // Base case
    if (root == NULL || (root->left == NULL &&
                         root->right == NULL))
        return;

    // If left child is smaller than current result,
    // update result and recur for left subarray.
    if (res > arr[root->left->idx] &&
       root->left->idx != root->idx)
    {
        res = arr[root->left->idx];
        traverseHeight(root->right, arr, res);
    }

    // If right child is smaller than current result,
    // update result and recur for left subarray.
    else if (res > arr[root->right->idx] &&
             root->right->idx != root->idx)
    {
        res = arr[root->right->idx];
        traverseHeight(root->left, arr, res);
    }
}

// Prints minimum and second minimum in arr[0..n-1]
void findSecondMin(int arr[], int n)
{
    // Create a list to store nodes of current
    // level
    list<Node *> li;

    Node *root = NULL;
    for (int i = 0; i < n; i += 2)
    {
        Node *t1 = createNode(i);
```

```
    Node *t2 = NULL;
    if (i + 1 < n)
    {
        // Make a node for next element
        t2 = createNode(i + 1);

        // Make smaller of two as root
        root = (arr[i] < arr[i + 1])? createNode(i) :
                                      createNode(i + 1);

        // Make two nodes as children of smaller
        root->left = t1;
        root->right = t2;

        // Add root
        li.push_back(root);
    }
    else
        li.push_back(t1);
}

int lsize = li.size();

// Construct the complete tournament tree from above
// prepared list of winners in first round.
while (lsize != 1)
{
    // Find index of last pair
    int last = (lsize & 1)? (lsize - 2) : (lsize - 1);

    // Process current list items in pair
    for (int i = 0; i < last; i += 2)
    {
        // Extract two nodes from list, make a new
        // node for winner of two
        Node *f1 = li.front();
        li.pop_front();

        Node *f2 = li.front();
        li.pop_front();
        root = (arr[f1->idx] < arr[f2->idx])?
            createNode(f1->idx) : createNode(f2->idx);

        // Make winner as parent of two
        root->left = f1;
        root->right = f2;

        // Add winner to list of next level
```

```
            li.push_back(root);
        }
        if (lsize & 1)
        {
            li.push_back(li.front());
            li.pop_front();
        }
        lsize = li.size();
    }

    // Traverse tree from root to find second minimum
    // Note that minimum is already known and root of
    // tournament tree.
    int res = INT_MAX;
    traverseHeight(root, arr, res);
    cout << "Minimum: " << arr[root->idx]
        << ", Second minimum: " << res << endl;
}

// Driver code
int main()
{
    int arr[] = {61, 6, 100, 9, 10, 12, 17};
    int n = sizeof(arr)/sizeof(arr[0]);
    findSecondMin(arr, n);
    return 0;
}
```

Output:

```
Minimum: 6, Second minimum: 9
```

We can optimize above code by avoid creation of leaf nodes, as that information is stored in the array itself (and we know that the elements were compared in pairs).

Source

https://www.geeksforgeeks.org/second-minimum-element-using-minimum-comparisons/

Chapter 237

Shortest Un-ordered Subarray

Shortest Un-ordered Subarray - GeeksforGeeks

An array is given of n length, and problem is that we have to find the length of shortest unordered {neither increasing nor decreasing} sub array in given array.

Examples:

```
Input : n = 5
        7 9 10 8 11
Output : 3
Explanation : 9 10 8 unordered sub array.

Input : n = 5
       1 2 3 4 5
Output : 0
Explanation :  Array is in increasing order.
```

The idea is based on the fact that size of shortest subarray would be either 0 or 3. We have to check array element is either increasing or decreasing, if all array elements are in increasing or decreasing, then length of shortest sub array is 0, And if either the array element is not follow the increasing or decreasing then it shortest length is 3.

C++

```
 // CPP program to find shortest subarray which is
// unsorted.
#include <bits/stdc++.h>
using namespace std;

// bool function for checking an array elements
// are in increasing.
bool increasing(int a[], int n)
```

```
{
    for (int i = 0; i < n - 1; i++)
        if (a[i] >= a[i + 1])
            return false;
    return true;
}

// bool function for checking an array
// elements are in decreasing.
bool decreasing(int a[], int n)
{
    for (int i = 0; i < n - 1; i++)
        if (a[i] < a[i + 1])
            return false;
    return true;
}

int shortestUnsorted(int a[], int n)
{
    // increasing and decreasing are two functions.
    // if function return true value then print
    // 0 otherwise 3.
    if (increasing(a, n) == true ||
       decreasing(a, n) == true)
        return 0;
    else
        return 3;
}

// Driver code
int main()
{
    int ar[] = { 7, 9, 10, 8, 11 };
    int n = sizeof(ar) / sizeof(ar[0]);
    cout << shortestUnsorted(ar, n);
    return 0;
}
```

Java

```
 // JAVA program to find shortest subarray which is
// unsorted.
import java.util.*;
import java.io.*;

class GFG {

    // boolean function to check array elements
```

```
    // are in increasing order or not
    public static boolean increasing(int a[],int n)
    {
        for (int i = 0; i < n - 1; i++)
            if (a[i] >= a[i + 1])
                return false;

        return true;
    }

    // boolean function to check array elements
    // are in decreasing order or not
    public static boolean decreasing(int arr[],int n)
    {
        for (int i = 0; i < n - 1; i++)
            if (arr[i] < arr[i + 1])
                return false;

        return true;
    }

    public static int shortestUnsorted(int a[],int n)
    {

        // increasing and decreasing are two functions.
        // if function return true value then print
        // 0 otherwise 3.
        if (increasing(a, n) == true ||
                            decreasing(a, n) == true)
            return 0;
        else
            return 3;
    }

    // driver program
    public static void main (String[] args) {

        int ar[] = new int[]{7, 9, 10, 8, 11};
        int n = ar.length;

        System.out.println(shortestUnsorted(ar,n));
    }
}

// This code is contributed by Akash Singh.
```

Python3

```
 # Python3 program to find shortest
# subarray which is unsorted

# Bool function for checking an array
# elements are in increasing
def increasing(a, n):

    for i in range(0, n - 1):
        if (a[i] >= a[i + 1]):
            return False

    return True

# Bool function for checking an array
# elements are in decreasing
def decreasing(a, n):

    for i in range(0, n - 1):
        if (a[i] < a[i + 1]):
            return False

    return True

def shortestUnsorted(a, n):

    # increasing and decreasing are two functions.
    # if function return True value then print
    # 0 otherwise 3.
    if (increasing(a, n) == True or
        decreasing(a, n) == True):
        return 0
    else:
        return 3

# Driver code
ar = [7, 9, 10, 8, 11]
n = len(ar)
print(shortestUnsorted(ar, n))

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
 // Program to find the shortest
// subarray which is unsorted.
using System;

class GFG {
```

```
    // boolean function to check
    // array elements are in the
    // increasing order or not
    public static bool increasing(int[] a, int n)
    {
        for (int i = 0; i < n - 1; i++)
            if (a[i] >= a[i + 1])
                return false;

        return true;
    }

    // boolean function to check
    // array elements are in the
    // decreasing order or not
    public static bool decreasing(int[] arr, int n)
    {
        for (int i = 0; i < n - 1; i++)
            if (arr[i] < arr[i + 1])
                return false;

        return true;
    }

    public static int shortestUnsorted(int[] a, int n)
    {

        // increasing and decreasing are
        // two functions. function return
        // true value then print 0 else 3
         if (increasing(a, n) == true ||
             decreasing(a, n) == true)
            return 0;
        else
            return 3;
    }

    // Driver program
    public static void Main()
    {

        int[] ar = new int[] { 7, 9, 10, 8, 11 };
        int n = ar.Length;
        Console.WriteLine(shortestUnsorted(ar, n));
    }
}
```

```
// This code is contributed by vt_m.
```

PHP

```
 <?php
// php program to find shortest
// subarray which is unsorted.

// bool function for checking an
// array elements are in increasing.
function increasing($a, $n)
{
    for ( $i = 0; $i < $n - 1; $i++)
        if ($a[$i] >= $a[$i + 1])
            return false;

    return true;
}

// bool function for checking an
// array elements are in decreasing.
function decreasing($a, $n)
{
    for ($i = 0; $i < $n - 1; $i++)
        if ($a[$i] < $a[$i + 1])
            return false;
    return true;
}

function shortestUnsorted($a, $n)
{

    // increasing and decreasing are
    // two functions. if function
    // return true value then print
    // 0 otherwise 3.
    if (increasing($a, $n) == true ||
          decreasing($a, $n) == true)
        return 0;
    else
        return 3;
}

// Driver code
$ar = array( 7, 9, 10, 8, 11 );
$n = sizeof($ar);
echo shortestUnsorted($ar, $n);
```

```
// This code is contributed by
// nitin mittal.
?>
```

Output :

```
3
```

Improved By : nitin mittal

Source

https://www.geeksforgeeks.org/shortest-un-ordered-subarray/

Chapter 238

Smallest Difference Triplet from Three arrays

Smallest Difference Triplet from Three arrays - GeeksforGeeks

Three arrays of same size are given. Find a triplet such that maximum – minimum in that triplet is minimum of all the triplets. A triplet should be selected in a way such that it should have one number from each of the three given arrays.

If there are 2 or more smallest difference triplets, then the one with the smallest sum of its elements should be displayed.

Examples :

```
Input : arr1[] = {5, 2, 8}
    arr2[] = {10, 7, 12}
    arr3[] = {9, 14, 6}
Output : 8, 7, 6

Input : arr1[] = {15, 12, 18, 9}
    arr2[] = {10, 17, 13, 8}
    arr3[] = {14, 16, 11, 5}
Output : 11, 10, 9
```

Note: The elements of the triplet are displayed in non-decreasing order.

Simple Solution : Consider each an every triplet and find the required smallest difference triplet out of them. Complexity of $O(n^3)$.

Efficient Solution:

1. Sort the 3 arrays in non-decreasing order.
2. Start three pointers from left most elements of three arrays.

3. Now find min and max and calculate max-min from these three elements.
4. Now increment pointer of minimum element's array.
5. Repeat steps 2, 3, 4, for the new set of pointers until any one pointer reaches to its end.

C++

```
 // C++ implementation of smallest difference triplet
#include <bits/stdc++.h>
using namespace std;

// function to find maximum number
int maximum(int a, int b, int c)
{
   return max(max(a, b), c);
}

// function to find minimum number
int minimum(int a, int b, int c)
{
   return min(min(a, b), c);
}

// Finds and prints the smallest Difference Triplet
void smallestDifferenceTriplet(int arr1[], int arr2[],
                                    int arr3[], int n)
{
    // sorting all the three arrays
    sort(arr1, arr1+n);
    sort(arr2, arr2+n);
    sort(arr3, arr3+n);

    // To store resultant three numbers
    int res_min, res_max, res_mid;

    // pointers to arr1, arr2, arr3
    // respectively
    int i = 0, j = 0, k = 0;

    // Loop until one array reaches to its end
    // Find the smallest difference.
    int diff = INT_MAX;
    while (i < n && j < n && k < n)
    {
        int sum = arr1[i] + arr2[j] + arr3[k];

        // maximum number
        int max = maximum(arr1[i], arr2[j], arr3[k]);
```

```
        // Find minimum and increment its index.
        int min = minimum(arr1[i], arr2[j], arr3[k]);
        if (min == arr1[i])
            i++;
        else if (min == arr2[j])
            j++;
        else
            k++;

        // comparing new difference with the
        // previous one and updating accordingly
        if (diff > (max-min))
        {
            diff = max - min;
            res_max = max;
            res_mid = sum - (max + min);
            res_min = min;
        }
    }

    // Print result
    cout << res_max << ", " << res_mid << ", " << res_min;
}

// Driver program to test above
int main()
{
    int arr1[] = {5, 2, 8};
    int arr2[] = {10, 7, 12};
    int arr3[] = {9, 14, 6};
    int n = sizeof(arr1) / sizeof(arr1[0]);
    smallestDifferenceTriplet(arr1, arr2, arr3, n);
    return 0;
}
```

Java

```
 // Java implementation of smallest difference
// triplet
import java.util.Arrays;

class GFG {

    // function to find maximum number
    static int maximum(int a, int b, int c)
    {
        return Math.max(Math.max(a, b), c);
```

```
}

// function to find minimum number
static int minimum(int a, int b, int c)
{
    return Math.min(Math.min(a, b), c);
}

// Finds and prints the smallest Difference
// Triplet
static void smallestDifferenceTriplet(int arr1[],
                  int arr2[], int arr3[], int n)
{

    // sorting all the three arrays
    Arrays.sort(arr1);
    Arrays.sort(arr2);
    Arrays.sort(arr3);

    // To store resultant three numbers
    int res_min=0, res_max=0, res_mid=0;

    // pointers to arr1, arr2, arr3
    // respectively
    int i = 0, j = 0, k = 0;

    // Loop until one array reaches to its end
    // Find the smallest difference.
    int diff = 2147483647;

    while (i < n && j < n && k < n)
    {
        int sum = arr1[i] + arr2[j] + arr3[k];

        // maximum number
        int max = maximum(arr1[i], arr2[j], arr3[k]);

        // Find minimum and increment its index.
        int min = minimum(arr1[i], arr2[j], arr3[k]);
        if (min == arr1[i])
            i++;
        else if (min == arr2[j])
            j++;
        else
            k++;

        // comparing new difference with the
        // previous one and updating accordingly
```

```
            if (diff > (max - min))
            {
                diff = max - min;
                res_max = max;
                res_mid = sum - (max + min);
                res_min = min;
            }
        }

        // Print result
        System.out.print(res_max + ", " + res_mid
                                 + ", " + res_min);
    }

    //driver code
    public static void main (String[] args)
    {

        int arr1[] = {5, 2, 8};
        int arr2[] = {10, 7, 12};
        int arr3[] = {9, 14, 6};

        int n = arr1.length;

        smallestDifferenceTriplet(arr1, arr2, arr3, n);
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 implementation of smallest
# difference triplet

# Function to find maximum number
def maximum(a, b, c):
    return max(max(a, b), c)

# Function to find minimum number
def minimum(a, b, c):
    return min(min(a, b), c)

# Finds and prints the smallest
# Difference Triplet
def smallestDifferenceTriplet(arr1, arr2, arr3, n):

    # sorting all the three arrays
```

```
    arr1.sort()
    arr2.sort()
    arr3.sort()

    # To store resultant three numbers
    res_min = 0; res_max = 0; res_mid = 0

    # pointers to arr1, arr2,
    # arr3 respectively
    i = 0; j = 0; k = 0

    # Loop until one array reaches to its end
    # Find the smallest difference.
    diff = 2147483647
    while (i < n and j < n and k < n):

        sum = arr1[i] + arr2[j] + arr3[k]

        # maximum number
        max = maximum(arr1[i], arr2[j], arr3[k])

        # Find minimum and increment its index.
        min = minimum(arr1[i], arr2[j], arr3[k])
        if (min == arr1[i]):
            i += 1
        elif (min == arr2[j]):
            j += 1
        else:
            k += 1

        # Comparing new difference with the
        # previous one and updating accordingly
        if (diff > (max - min)):

            diff = max - min
            res_max = max
            res_mid = sum - (max + min)
            res_min = min

    # Print result
    print(res_max, ",", res_mid, ",", res_min)

# Driver code
arr1 = [5, 2, 8]
arr2 = [10, 7, 12]
arr3 = [9, 14, 6]
n = len(arr1)
smallestDifferenceTriplet(arr1, arr2, arr3, n)
```

```
# This code is contributed by Anant Agarwal.
```

C#

```
 // C# implementation of smallest
// difference triplet
using System;

class GFG
{

    // function to find
    // maximum number
    static int maximum(int a, int b, int c)
    {
        return Math.Max(Math.Max(a, b), c);
    }

    // function to find
    // minimum number
    static int minimum(int a, int b, int c)
    {
        return Math.Min(Math.Min(a, b), c);
    }

    // Finds and prints the
    // smallest Difference Triplet
    static void smallestDifferenceTriplet(int []arr1,
                                          int []arr2,
                                          int []arr3,
                                          int n)
    {

        // sorting all the
        // three arrays
        Array.Sort(arr1);
        Array.Sort(arr2);
        Array.Sort(arr3);

        // To store resultant
        // three numbers
        int res_min = 0, res_max = 0, res_mid = 0;

        // pointers to arr1, arr2,
        // arr3 respectively
        int i = 0, j = 0, k = 0;
```

```
    // Loop until one array
    // reaches to its end
    // Find the smallest difference.
    int diff = 2147483647;

    while (i < n && j < n && k < n)
    {
        int sum = arr1[i] +
                  arr2[j] + arr3[k];

        // maximum number
        int max = maximum(arr1[i],
                          arr2[j], arr3[k]);

        // Find minimum and
        // increment its index.
        int min = minimum(arr1[i],
                          arr2[j], arr3[k]);
        if (min == arr1[i])
            i++;
        else if (min == arr2[j])
            j++;
        else
            k++;

        // comparing new difference
        // with the previous one and
        // updating accordingly
        if (diff > (max - min))
        {
            diff = max - min;
            res_max = max;
            res_mid = sum - (max + min);
            res_min = min;
        }
    }

    // Print result
    Console.WriteLine(res_max + ", " +
                      res_mid + ", " +
                      res_min);
}

// Driver code
static public void Main ()
{
    int []arr1 = {5, 2, 8};
    int []arr2 = {10, 7, 12};
```

```
        int []arr3 = {9, 14, 6};

        int n = arr1.Length;

        smallestDifferenceTriplet(arr1, arr2,
                                  arr3, n);
    }
}

// This code is contributed by ajit.
```

PHP

```
 <?php
// PHP implementation of
// smallest difference triplet

// function to find
// maximum number
function maximum($a, $b, $c)
{
    return max(max($a, $b), $c);
}

// function to find
// minimum number
function minimum($a, $b, $c)
{
    return min(min($a, $b), $c);
}

// Finds and prints the
// smallest Difference Triplet
function smallestDifferenceTriplet($arr1, $arr2,
                                   $arr3, $n)
{
    // sorting all the
    // three arrays
    sort($arr1);
    sort($arr2);
    sort($arr3);

    // To store resultant
    // three numbers
    $res_min; $res_max; $res_mid;

    // pointers to arr1, arr2,
    // arr3 respectively
```

```
    $i = 0; $j = 0; $k = 0;

    // Loop until one array reaches
    // to its end
    // Find the smallest difference.
    $diff = PHP_INT_MAX;
    while ($i < $n && $j < $n && $k < $n)
    {
        $sum = $arr1[$i] +
               $arr2[$j] +
               $arr3[$k];

        // maximum number
        $max = maximum($arr1[$i],
                       $arr2[$j],
                       $arr3[$k]);

        // Find minimum and
        // increment its index.
        $min = minimum($arr1[$i],
                       $arr2[$j],
                       $arr3[$k]);
        if ($min == $arr1[$i])
            $i++;
        else if ($min == $arr2[$j])
            $j++;
        else
            $k++;

        // comparing new difference
        // with the previous one
        // and updating accordingly
        if ($diff > ($max - $min))
        {
            $diff = $max - $min;
            $res_max = $max;
            $res_mid = $sum - ($max + $min);
            $res_min = $min;
        }
    }

    // Print result
    echo $res_max , ", " ,
         $res_mid , ", " ,
         $res_min;
}

// Driver Code
```

```
$arr1 = array(5, 2, 8);
$arr2 = array(10, 7, 12);
$arr3 = array(9, 14, 6);

$n = sizeof($arr1);
smallestDifferenceTriplet($arr1, $arr2,
                          $arr3, $n);

// This code is contributed by ajit
?>
```

Output :

```
7, 6, 5
```

Time Complexity : O(n log n)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/smallest-difference-triplet-from-three-arrays/

Chapter 239

Smallest greater elements in whole array

Smallest greater elements in whole array - GeeksforGeeks

An array is given of n length, and we need to calculate the next greater element for each element in given array. If next greater element is not available in given array then we need to fill '_' at that index place.

Examples :

```
Input :  6 3 9 8 10 2 1 15 7
Output : 7 6 10 9 15 3 2 _ 8
Here every element of array has next greater
element but at index 7,
15 is the greatest element of given array
and no other element is greater from 15
so at the index of 15 we fill with '_' .

Input  : 13 6 7 12
Output : _ 7 12 13
Here, at index 0, 13 is the greatest
value in given array and no other
array element is greater from 13 so
at index 0 we fill '_'.
```

Asked in : Zoho

A **simple solution** is to use two loops nested. The outer loop picks all elements one by one and inner loop finds the next greater element by linearly searching from beginning to end.

C++

```
 // Simple CPP program to find smallest
// greater element in whole array for
// every element.
#include <bits/stdc++.h>
using namespace std;

void smallestGreater(int arr[], int n)
{
    for (int i = 0; i < n; i++) {

        // Find the closest greater element
        // for arr[j] in the entire array.
        int diff = INT_MAX, closest = -1;
        for (int j = 0; j < n; j++) {
            if ( arr[i] < arr[j] &&
                 arr[j] - arr[i] < diff)
            {
                diff = arr[j] - arr[i];
                closest = j;
            }
        }

        // Check if arr[i] is largest
        (closest == -1)?  cout << "_ "  :
              cout << arr[closest] << " ";
     }
}

// Driver code
int main()
{
    int ar[] = { 6, 3, 9, 8, 10, 2, 1, 15, 7 };
    int n = sizeof(ar) / sizeof(ar[0]);
    smallestGreater(ar, n);
    return 0;
}
```

Java

```
 // Simple Java program to find
// smallest greater element in
// whole array for every element.
import java.io.*;

class GFG
{
static void smallestGreater(int arr[],
                            int n)
```

```
{
    for (int i = 0; i < n; i++)
    {

        // Find the closest greater
        // element for arr[j] in
        // the entire array.
        int diff = Integer.MAX_VALUE;
        int closest = -1;
        for (int j = 0; j < n; j++)
        {
            if (arr[i] < arr[j] &&
                arr[j] - arr[i] < diff)
            {
                diff = arr[j] - arr[i];
                closest = j;
            }
        }

        // Check if arr[i] is largest
        if(closest == -1)
        System.out.print( "_ " );
        else
        System.out.print(arr[closest] + " ");
    }
}

// Driver code
public static void main (String[] args)
{
    int ar[] = {6, 3, 9, 8, 10,
                2, 1, 15, 7};
    int n = ar.length;
    smallestGreater(ar, n);
}
}

// This code is contributed by anuj_67.
```

Python3

```
 # Simple Python program to find smallest
# greater element in whole array for
# every element.
def smallestGreater(arr, n) :
    for i in range(0, n) :

        # Find the closest greater element
```

```
        # for arr[j] in the entire array.
        diff = 1000;
        closest = -1;
        for j in range(0, n) :
            if ( arr[i] < arr[j] and
                  arr[j] - arr[i] < diff) :
                diff = arr[j] - arr[i];
                closest = j;

        # Check if arr[i] is largest
        if (closest == -1) :
            print ("_ ", end = "");
        else :
            print ("{} ".format(arr[closest]),
                                    end = "");

# Driver code
ar = [6, 3, 9, 8, 10, 2, 1, 15, 7];
n = len(ar) ;
smallestGreater(ar, n);

# This code is contributed by Manish Shaw
# (manishshaw1)
```

C#

```
 // Simple C# program to find
// smallest greater element in
// whole array for every element.
using System;

class GFG
{
static void smallestGreater(int []arr,
                            int n)
{
    for (int i = 0; i < n; i++)
    {

        // Find the closest greater
        // element for arr[j] in
        // the entire array.
        int diff = int.MaxValue;
        int closest = -1;
        for (int j = 0; j < n; j++)
        {
            if (arr[i] < arr[j] &&
                arr[j] - arr[i] < diff)
```

```
            {
                diff = arr[j] - arr[i];
                closest = j;
            }
        }

        // Check if arr[i] is largest
        if(closest == -1)
        Console.Write( "_ " );
        else
        Console.Write(arr[closest] + " ");
    }
}

// Driver code
public static void Main()
{
    int []ar = {6, 3, 9, 8, 10,
                2, 1, 15, 7};
    int n = ar.Length;
    smallestGreater(ar, n);
}
}

// This code is contributed by anuj_67.
```

PHP

```
 <?php
// Simple PHP program to find smallest
// greater element in whole array for
// every element.

function smallestGreater($arr, $n)
{
    for ( $i = 0; $i < $n; $i++) {

        // Find the closest greater element
        // for arr[j] in the entire array.
        $diff = PHP_INT_MAX; $closest = -1;
        for ( $j = 0; $j < $n; $j++) {
            if ( $arr[$i] < $arr[$j] &&
                $arr[$j] - $arr[$i] < $diff)
            {
                $diff = $arr[$j] - $arr[$i];
                $closest = $j;
            }
        }
```

```
        // Check if arr[i] is largest
        if ($closest == -1)
        echo "_ " ;
        else
            echo $arr[$closest] , " ";
    }
}

    // Driver code
    $ar = array (6, 3, 9, 8, 10, 2, 1, 15, 7);
    $n = sizeof($ar) ;
    smallestGreater($ar, $n);

// This code is contributed by ajit
?>
```

Output:

```
7 6 10 9 15 3 2 _ 8
```

Time Complexity : O(n*n)
Auxiliary Space: O(1)

An **efficient solution** is to one by one insert elements in a set (A self balancing binary search tree). After inserting into set, we search elements. After we find iterator of the searched element, we move iterator to next (note that set stores elements in sorted order) to find element which is just greater.

C++

```
 // Efficient CPP program to find smallest
// greater element in whole array for
// every element.
#include <bits/stdc++.h>
using namespace std;

void smallestGreater(int arr[], int n)
{
    set<int> s;
    for (int i = 0; i < n; i++)
       s.insert(arr[i]);

    for (int i = 0; i < n; i++)
    {
       auto it = s.find(arr[i]);
       it++;
```

```
        if (it != s.end())
            cout << *it << " ";
        else
            cout << "_ ";
      }
}

// Driver code
int main()
{
    int ar[] = { 6, 3, 9, 8, 10, 2, 1, 15, 7 };
    int n = sizeof(ar) / sizeof(ar[0]);
    smallestGreater(ar, n);
    return 0;
}
```

Output :

```
7 6 10 9 15 3 2 _ 8
```

Time Complexity : O(n Log n). Note that self balancing search tree (implemented by set in C++) insert operations take O(Log n) time to insert and find.
Auxiliary Space: O(n)

We can also use sorting followed bybinary searches to solve the above problem in same time and same auxiliary space.

Improved By : jit_t, vt_m, manishshaw1

Source

https://www.geeksforgeeks.org/smallest-greater-elements-in-whole-array/

Chapter 240

Smallest number whose set bits are maximum in a given range

Smallest number whose set bits are maximum in a given range - GeeksforGeeks

Given a positive integer '**l**' and '**r**'. Find the smallest number '**n**' such that **l <= n <= r** and count of number of set bits(number of '1's in binary representation) is maximum as possible.
Examples :

```
Input: 1 4
Output: 3
Explanation:
Binary representation from '1' to '4':
110 = 0012
210 = 0102
310 = 0112
110 = 1002
Thus number '3' has maximum set bits = 2

Input: 1 10
Output: 7
```

Simple approach is to traverse from 'l' to 'r' and count the set bits for each 'x'(l <= n <= r) and print the number whose count is maximum among them. Time complexity of this approach is O(n*log(r)).

C++

```
// C++ program to find number whose set
```

```
// bits are maximum among 'l' and 'r'
#include <bits/stdc++.h>
using namespace std;

// Returns smallest number whose set bits
// are maximum in given range.
int countMaxSetBits(int left, int right)
{
    // Initialize the maximum count and
    // final answer as 'num'
    int max_count = -1, num;
    for (int i = left; i <= right; ++i) {
        int temp = i, cnt = 0;

        // Traverse for every bit of 'i'
        // number
        while (temp) {
            if (temp & 1)
                ++cnt;
            temp >>= 1;
        }

        // If count is greater than previous
        // calculated max_count, update it
        if (cnt > max_count) {
            max_count = cnt;
            num = i;
        }
    }
    return num;
}

// Driver code
int main()
{
    int l = 1, r = 5;
    cout << countMaxSetBits(l, r) << "\n";

    l = 1, r = 10;
    cout << countMaxSetBits(l, r);
    return 0;
}
```

Python 3

```
 # Python code to find number whose set
# bits are maximum among 'l' and 'r'
```

```
def countMaxSetBits( left, right):
    max_count = -1
    for i in range(left, right+1):
        temp = i
        cnt = 0

        # Traverse for every bit of 'i'
        # number
        while temp:
            if temp & 1:
                cnt +=1
            temp = temp >> 1

        # If count is greater than previous
        # calculated max_count, update it
        if cnt > max_count:
            max_count = cnt
            num=i
    return num

# driver code
l = 1
r = 5
print(countMaxSetBits(l, r))
l = 1
r = 10
print(countMaxSetBits(l, r))

# This code is contributed by "Abhishek Sharma 44"
```

PHP

```
 <?php
// PHP program to find number
// whose set bits are maximum
// among 'l' and 'r'

// Returns smallest number
// whose set bits are maximum
// in given range.

function countMaxSetBits($left, $right)
{
    // Initialize the maximum
    // count and final answer
    // as 'num'
    $max_count = -1; $num;
    for ($i = $left; $i <= $right; ++$i)
```

```
    {
        $temp = $i; $cnt = 0;

        // Traverse for every
        // bit of 'i' number
        while ($temp)
        {
            if ($temp & 1)
                ++$cnt;
            $temp >>= 1;
        }

        // If count is greater than
        // previous calculated
        // max_count, update it
        if ($cnt > $max_count)
        {
            $max_count = $cnt;
            $num = $i;
        }
    }
    return $num;
}

// Driver code
$l = 1; $r = 5;
echo countMaxSetBits($l, $r), "\n";

$l = 1; $r = 10;
echo countMaxSetBits($l, $r);

// This code is contributed by m_kit
?>
```

Output :

```
3
7
```

Efficient approach is to use bit-manipulation. Instead of iterating for every number from 'l' to 'r', iterate only after updating the desired number('num') i.e., take the bitwise 'OR' of number with the consecutive number. For instance,

```
Let l = 2, and r = 10
1. num = 2
2. x = num OR (num + 1)
```

```
       = 2 | 3 = 010 | 011 = 011
   num = 3(011)
3. x = 3 | 4 = 011 | 100 = 111
   num = 7(111)
4. x = 7 | 8 = 0111 | 1000 = 1111
   Since 15(11112) is greater than
   10, thus stop traversing for next number.
5. Final answer = 7
```

C++

```
 // C++ program to find number whose set
// bits are maximum among 'l' and 'r'
#include <bits/stdc++.h>
using namespace std;

// Returns smallest number whose set bits
// are maximum in given range.
int countMaxSetBits(int left, int right)
{
    while ((left | (left + 1)) <= right)
        left |= left + 1;

    return left;
}

// Driver code
int main()
{
    int l = 1, r = 5;
    cout << countMaxSetBits(l, r) << "\n";

    l = 1, r = 10;
    cout << countMaxSetBits(l, r) ;
    return 0;
}
```

Java

```
 // Java program to find number
// whose set bits are maximum
// among 'l' and 'r'
import java.io.*;

class GFG
{
```

```
    // Returns smallest number
    // whose set bits are
    // maximum in given range.
    static int countMaxSetBits(int left,
                               int right)
    {
    while ((left | (left + 1)) <= right)
        left |= left + 1;

    return left;
    }

// Driver code
public static void main (String[] args)
{
    int l = 1;
    int r = 5;
    System.out.println(countMaxSetBits(l, r));

    l = 1;
    r = 10;
    System.out.println(countMaxSetBits(l, r));
}
}

// This code is contributed by @ajit
```

Python3

```
 # Python code to find number whose set
# bits are maximum among 'l' and 'r'

def countMaxSetBits( left, right):

    while(left | (left+1)) <= right:
        left |= left+1
    return left

# driver code
l = 1
r = 5
print(countMaxSetBits(l, r))
l = 1
r = 10
print(countMaxSetBits(l, r))

# This code is contributed by "Abhishek Sharma 44"
```

C#

```
 // C# program to find number
// whose set bits are maximum
// among 'l' and 'r'
using System;

class GFG
{

    // Returns smallest number
    // whose set bits are
    // maximum in given range.
    static int countMaxSetBits(int left,
                               int right)
    {
    while ((left | (left + 1)) <= right)
        left |= left + 1;

    return left;
    }

// Driver code
static public void Main ()
{
    int l = 1;
    int r = 5;
    Console.WriteLine(countMaxSetBits(l, r));

    l = 1;
    r = 10;
    Console.WriteLine(countMaxSetBits(l, r));
}
}

// This code is contributed by @ajit
```

PHP

```
 <?php
// PHP program to find number
// whose set bits are maximum
// among 'l' and 'r'

// Returns smallest number
// whose set bits are
// maximum in given range.
```

```
function countMaxSetBits($left,
                         $right)
{
    while (($left | ($left + 1)) <= $right)
        $left |= $left + 1;

    return $left;
}

// Driver code
$l = 1 ; $r = 5;
echo countMaxSetBits($l, $r) , "\n";

$l = 1; $r = 10;
echo countMaxSetBits($l, $r) ;

// This code is contributed by aj_36
?>
```

Output :

```
3
7
```

Time complexity: O(log(n))
Auxiliary space: O(1)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/smallest-number-whose-set-bits-maximum-given-range/

Chapter 241

Smallest number with at least n trailing zeroes in factorial

Smallest number with at least n trailing zeroes in factorial - GeeksforGeeks

Given a number **n**. The task is to find the smallest number whose factorial contains at least n trailing zeroes.

Examples :

```
Input : n = 1
Output : 5
1!, 2!, 3!, 4! does not contain trailing zero.
5! = 120, which contains one trailing zero.

Input : n = 6
Output : 25
```

In the article for Count trailing zeroes in factorial of a number, we have discussed number of zeroes is equal to number of 5's in prime factors of x!. We have discussed below formula to count number of 5's.

```
Trailing 0s in x! = Count of 5s in prime factors of x!
                  = floor(x/5) + floor(x/25) + floor(x/125) + ....
```

Let us take few examples to observe pattern

```
5!  has 1 trailing zeroes
[All numbers from 6 to 9
```

```
 have 1 trailing zero]

10! has 2 trailing zeroes
[All numbers from 11 to 14
 have 2 trailing zeroes]

15! to 19! have 3 trailing zeroes

20! to 24! have 4 trailing zeroes

25! to 29! have 6 trailing zeroes
```

We can notice that, the minimum value whose factorial contain n trailing zeroes is 5*n.

So, to find minimum value whose factorial contains n trailing zeroes, use binary search on range from 0 to 5*n. And, find the smallest number whose factorial contains n trailing zeroes.

C++

```
 // C++ program tofind smallest number whose
// factorial contains at least n trailing
// zeroes.
#include<bits/stdc++.h>
using namespace std;

// Return true if number's factorial contains
// at least n trailing zero else false.
bool check(int p, int n)
{
    int temp = p, count = 0, f = 5;
    while (f <= temp)
    {
        count += temp/f;
        f = f*5;
    }
    return (count >= n);
}

// Return smallest number whose factorial
// contains at least n trailing zeroes
int findNum(int n)
{
    // If n equal to 1, return 5.
    // since 5! = 120.
    if (n==1)
        return 5;

    // Initalising low and high for binary
```

```
    // search.
    int low = 0;
    int high = 5*n;

    // Binary Search.
    while (low <high)
    {
        int mid = (low + high) >> 1;

        // Checking if mid's factorial contains
        // n trailing zeroes.
        if (check(mid, n))
            high = mid;
        else
            low = mid+1;
    }

    return low;
}

// driver code
int main()
{
    int n = 6;
    cout << findNum(n) << endl;
    return 0;
}
```

Java

```
 // Java program tofind smallest number whose
// factorial contains at least n trailing
// zeroes.

class GFG
{
    // Return true if number's factorial contains
    // at least n trailing zero else false.
    static boolean check(int p, int n)
    {
        int temp = p, count = 0, f = 5;
        while (f <= temp)
        {
            count += temp / f;
            f = f * 5;
        }
        return (count >= n);
    }
```

```
    // Return smallest number whose factorial
    // contains at least n trailing zeroes
    static int findNum(int n)
    {
        // If n equal to 1, return 5.
        // since 5! = 120.
        if (n==1)
            return 5;

        // Initalising low and high for binary
        // search.
        int low = 0;
        int high = 5 * n;

        // Binary Search.
        while (low < high)
        {
            int mid = (low + high) >> 1;

            // Checking if mid's factorial
            // contains n trailing zeroes.
            if (check(mid, n))
                high = mid;
            else
                low = mid + 1;
        }

        return low;
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 6;
        System.out.println(findNum(n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 program tofind smallest
# number whose
# factorial contains at least
# n trailing zeroes
```

```
# Return true if number's factorial contains
# at least n trailing zero else false.
def check(p,n):

    temp = p
    count = 0
    f = 5
    while (f <= temp):
        count += temp/f
        f = f*5

    return (count >= n)

# Return smallest number whose factorial
# contains at least n trailing zeroes
def findNum(n):

    # If n equal to 1, return 5.
    # since 5! = 120.
    if (n==1):
        return 5

    # Initalizing low and high for binary
    # search.
    low = 0
    high = 5*n

    # Binary Search.
    while (low <high):

        mid = (low + high) >> 1

        # Checking if mid's factorial contains
        # n trailing zeroes.
        if (check(mid, n)):
            high = mid
        else:
            low = mid+1


    return low


# driver code
n = 6
print(findNum(n))

# This code is contributed
```

```
# by Anant Agarwal.
```

C#

```
 // C# program tofind smallest number whose
// factorial contains at least n trailing
// zeroes.
using System;

class GFG
{
    // Return true if number's factorial contains
    // at least n trailing zero else false.
    static bool check(int p, int n)
    {
        int temp = p, count = 0, f = 5;
        while (f <= temp)
        {
            count += temp / f;
            f = f * 5;
        }
        return (count >= n);
    }

    // Return smallest number whose factorial
    // contains at least n trailing zeroes
    static int findNum(int n)
    {
        // If n equal to 1, return 5.
        // since 5! = 120.
        if (n == 1)
            return 5;

        // Initalising low and high for binary
        // search.
        int low = 0;
        int high = 5 * n;

        // Binary Search.
        while (low < high)
        {
            int mid = (low + high) >> 1;

            // Checking if mid's factorial
            // contains n trailing zeroes.
            if (check(mid, n))
                high = mid;
            else
```

```
                low = mid + 1;
        }

        return low;
    }

    // Driver code
    public static void Main ()
    {
        int n = 6;

        Console.WriteLine(findNum(n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
 <?php
// PHP program to find smallest
// number whose factorial contains
// at least n trailing zeroes.

// Return true if number's
// factorial contains at
// least n trailing zero
// else false.
function check($p, $n)
{
    $temp = $p; $count = 0; $f = 5;
    while ($f <= $temp)
    {
        $count += $temp / $f;
        $f = $f * 5;
    }
    return ($count >= $n);
}

// Return smallest number
// whose factorial contains
// at least n trailing zeroes
function findNum($n)
{
    // If n equal to 1, return 5.
    // since 5! = 120.
    if ($n == 1)
        return 5;
```

```
    // Initalising low and high
    // for binary search.
    $low = 0;
    $high = 5 * $n;

    // Binary Search.
    while ($low < $high)
    {
        $mid = ($low + $high) >> 1;

        // Checking if mid's factorial
        // contains n trailing zeroes.
        if (check($mid, $n))
            $high = $mid;
        else
            $low = $mid + 1;
    }

    return $low;
}

// Driver Code
$n = 6;
echo(findNum($n));

// This code is contributed by Ajit.
?>
```

Output :

```
25
```

Improved By : jit_t

Source

https://www.geeksforgeeks.org/smallest-number-least-n-trailing-zeroes-factorial/

Chapter 242

Sort a Rotated Sorted Array

Sort a Rotated Sorted Array - GeeksforGeeks

You are given a rotated sorted array and your aim is to restore its original sort in place.

Expected to use O(1) extra space and O(n) time complexity.
Examples:

```
Input : [3, 4, 1, 2]
Output : [1, 2, 3, 4]

Input : [2, 3, 4, 1]
Output : [1, 2, 3, 4]
```

We find the point of rotation. Then we rotate array using reversal algorithm.

```
 1. First find the split point where the sorting breaks.
 2. Then call the reverse function in three steps.
     - From zero index to split index.
     - From split index to end index.
     - From zero index to end index.
```

```
 // C++ implementation for restoring original
// sort in rotated sorted array
#include <bits/stdc++.h>
using namespace std;

// Function to restore the Original Sort
void restoreSortedArray(int arr[], int n)
{
    for (int i = 0; i < n; i++) {
```

```
        if (arr[i] > arr[i + 1]) {

            // In reverse(), the first parameter
            // is iterator to beginning element
            // and second parameter is iterator
            // to last element plus one.
            reverse(arr, arr+i+1);
            reverse(arr + i + 1, arr + n);
            reverse(arr, arr + n);
        }
    }
}

// Function to print the Array
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
}

// Driver function
int main()
{
    int arr[] = { 3, 4, 5, 1, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    restoreSortedArray(arr, n);
    printArray(arr, n);
    return 0;
}
```

Output:

```
1 2 3 4 5
```

We can binary search to find the rotation point as discussed here .

Efficient code approach using binary search:

1. First find the index of minimum element (split index) in the array using binary search
2. Then call the reverse function in three steps.

- From zero index to split index.
- From split index to end index.
- From zero index to end index.

```
// C++ implementation for restoring original
```

```
// sort in rotated sorted array using binary search
#include <bits/stdc++.h>

using namespace std;

// Function to find start index of array
int findStartIndexOfArray(int arr[], int low,int high)
{
    if (low>high)
    {
        return -1;
    }

    if (low == high)
    {
        return low;
    }

    int mid = low + (high-low)/2;
    if(arr[mid] > arr[mid+1])
        return mid+1;

    if(arr[mid-1] > arr[mid])
        return mid;

    if(arr[low] > arr[mid])
        return findStartIndexOfArray(arr, low, mid-1);
    else
        return findStartIndexOfArray(arr, mid+1, high);
}

// Function to restore the Original Sort
void restoreSortedArray(int arr[], int n)
{
    // array is already sorted
    if (arr[0] < arr[n-1])
        return;

    int start = findStartIndexOfArray(arr, 0, n-1);
    // In reverse(), the first parameter
    // is iterator to beginning element
    // and second parameter is iterator
    // to last element plus one.
    reverse(arr, arr + start);
    reverse(arr + start, arr + n);
    reverse(arr, arr + n);

}
```

```
// Function to print the Array
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
}

// Driver function
int main()
{
    int arr[] = { 1, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    restoreSortedArray(arr, n);
    printArray(arr, n);
    return 0;
}
```

Output:

```
1 2 3 4 5
```

Improved By : Neha Jain

Source

https://www.geeksforgeeks.org/sort-rotated-sorted-array/

Chapter 243

Sort an array according to the order defined by another array

Sort an array according to the order defined by another array - GeeksforGeeks

Given two arrays A1[] and A2[], sort A1 in such a way that the relative order among the elements will be same as those are in A2. For the elements not present in A2, append them at last in sorted order.

```
Input: A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8}
       A2[] = {2, 1, 8, 3}
Output: A1[] = {2, 2, 1, 1, 8, 8, 3, 5, 6, 7, 9}
```

The code should handle all cases like number of elements in A2[] may be more or less compared to A1[]. A2[] may have some elements which may not be there in A1[] and vice versa is also possible.

Source: Amazon Interview | Set 110 (On-Campus)

Method 1 (Using Sorting and Binary Search)
Let size of A1[] be m and size of A2[] be n.
1) Create a temporary array temp of size m and copy contents of A1[] to it.
2) Create another array visited[] and initialize all entries in it as false. visited[] is used to mark those elements in temp[] which are copied to A1[].
3) Sort temp[]
4) Initialize the output index ind as 0.
5) Do following for every element of A2[i] in A2[]
.....a) Binary search for all occurrences of A2[i] in temp[], if present then copy all occurrences to A1[ind] and increment ind. Also mark the copied elements visited[]
6) Copy all unvisited elements from temp[] to A1[].

Time complexity: The steps 1 and 2 require O(m) time. Step 3 requires O(mLogm) time. Step 5 requires O(nLogm) time. Therefore overall time complexity is O(mLogm + nLogm).

Thanks to vivekfor suggesting this method. Following is the implementation of above algorithm.

C++

```
 // A C++ program to sort an array according to the order defined
// by another array
#include <iostream>
#include <algorithm>

using namespace std;

/* A Binary Search based function to find index of FIRST occurrence
  of x in arr[].  If x is not present, then it returns -1 */
int first(int arr[], int low, int high, int x, int n)
{
    if (high >= low)
    {
        int mid = low + (high-low)/2;  /* (low + high)/2; */
        if ((mid == 0 || x > arr[mid-1]) && arr[mid] == x)
            return mid;
        if (x > arr[mid])
            return first(arr, (mid + 1), high, x, n);
        return first(arr, low, (mid -1), x, n);
    }
    return -1;
}

// Sort A1[0..m-1] according to the order defined by A2[0..n-1].
void sortAccording(int A1[], int A2[], int m, int n)
{
    // The temp array is used to store a copy of A1[] and visited[]
    // is used mark the visited elements in temp[].
    int temp[m], visited[m];
    for (int i=0; i<m; i++)
    {
        temp[i] = A1[i];
        visited[i] = 0;
    }

    // Sort elements in temp
    sort(temp, temp + m);

    int ind = 0;  // for index of output which is sorted A1[]

    // Consider all elements of A2[], find them in temp[]
    // and copy to A1[] in order.
    for (int i=0; i<n; i++)
```

```
    {
        // Find index of the first occurrence of A2[i] in temp
        int f = first(temp, 0, m-1, A2[i], m);

        // If not present, no need to proceed
        if (f == -1) continue;

        // Copy all occurrences of A2[i] to A1[]
        for (int j = f; (j<m && temp[j]==A2[i]); j++)
        {
            A1[ind++] = temp[j];
            visited[j] = 1;
        }
    }

    // Now copy all items of temp[] which are not present in A2[]
    for (int i=0; i<m; i++)
        if (visited[i] == 0)
            A1[ind++] = temp[i];
}

// Utility function to print an array
void printArray(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver program to test above function.
int main()
{
    int A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8};
    int A2[] = {2, 1, 8, 3};
    int m = sizeof(A1)/sizeof(A1[0]);
    int n = sizeof(A2)/sizeof(A2[0]);
    cout << "Sorted array is \n";
    sortAccording(A1, A2, m, n);
    printArray(A1, m);
    return 0;
}
```

Java

```
 // A JAVA program to sort an array according
// to the order defined by another array
import java.io.*;
import java.util.Arrays;
```

```
class GFG {

    /* A Binary Search based function to find
    index of FIRST occurrence of x in arr[].
    If x is not present, then it returns -1 */
    static int first(int arr[], int low, int high,
                     int x, int n)
    {
        if (high >= low)
        {
            /* (low + high)/2; */
            int mid = low + (high-low)/2;

            if ((mid == 0 || x > arr[mid-1]) &&
                                arr[mid] == x)
                return mid;
            if (x > arr[mid])
                return first(arr, (mid + 1), high,
                                              x, n);
            return first(arr, low, (mid -1), x, n);
        }
        return -1;
    }

    // Sort A1[0..m-1] according to the order
    // defined by A2[0..n-1].
    static void sortAccording(int A1[], int A2[], int m,
                                                  int n)
    {
        // The temp array is used to store a copy
        // of A1[] and visited[] is used to mark the
        // visited elements in temp[].
        int temp[] = new int[m], visited[] = new int[m];
        for (int i = 0; i < m; i++)
        {
            temp[i] = A1[i];
            visited[i] = 0;
        }

        // Sort elements in temp
        Arrays.sort(temp);

        // for index of output which is sorted A1[]
        int ind = 0;

        // Consider all elements of A2[], find them
        // in temp[] and copy to A1[] in order.
```

```
        for (int i = 0; i < n; i++)
        {
            // Find index of the first occurrence
            // of A2[i] in temp
            int f = first(temp, 0, m-1, A2[i], m);

            // If not present, no need to proceed
            if (f == -1) continue;

            // Copy all occurrences of A2[i] to A1[]
            for (int j = f; (j < m && temp[j] == A2[i]);
                                                  j++)
            {
                A1[ind++] = temp[j];
                visited[j] = 1;
            }
        }

        // Now copy all items of temp[] which are
        // not present in A2[]
        for (int i = 0; i < m; i++)
            if (visited[i] == 0)
                A1[ind++] = temp[i];
    }

    // Utility function to print an array
    static void printArray(int arr[], int n)
    {
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver program to test above function.
    public static void main(String args[])
    {
        int A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8};
        int A2[] = {2, 1, 8, 3};
        int m = A1.length;
        int n = A2.length;
        System.out.println("Sorted array is ");
        sortAccording(A1, A2, m, n);
        printArray(A1, m);
    }
}


/*This code is contributed by Nikita Tiwari.*/
```

Python3

```
 """A Python 3  program to sort an array
according to the order defined by
another array"""

"""A Binary Search based function to find
index of FIRST occurrence of x in arr[].
If x is not present, then it returns -1 """

def first(arr, low, high, x, n) :
    if (high >= low) :
        mid = low + (high - low) // 2;  # (low + high)/2;
        if ((mid == 0 or x > arr[mid-1]) and arr[mid] == x) :
            return mid
        if (x > arr[mid]) :
            return first(arr, (mid + 1), high, x, n)
        return first(arr, low, (mid -1), x, n)

    return -1

# Sort A1[0..m-1] according to the order
# defined by A2[0..n-1].
def sortAccording(A1, A2, m, n) :

    """The temp array is used to store a copy
    of A1[] and visited[] is used mark the
    visited elements in temp[]."""
    temp = [0] * m
    visited = [0] * m

    for i in range(0, m) :
        temp[i] = A1[i]
        visited[i] = 0

    # Sort elements in temp
    temp.sort()

    # for index of output which is sorted A1[]
    ind = 0

    """Consider all elements of A2[], find
    them in temp[] and copy to A1[] in order."""
    for i in range(0,n) :

        # Find index of the first occurrence
        # of A2[i] in temp
        f = first(temp, 0, m-1, A2[i], m)
```

```
        # If not present, no need to proceed
        if (f == -1) :
            continue

        # Copy all occurrences of A2[i] to A1[]
        j = f
        while (j<m and temp[j]==A2[i]) :
            A1[ind] = temp[j];
            ind=ind+1
            visited[j] = 1
            j = j + 1

    # Now copy all items of temp[] which are
    # not present in A2[]
    for i in range(0, m) :
        if (visited[i] == 0) :
            A1[ind] = temp[i]
            ind = ind + 1

# Utility function to print an array
def printArray(arr, n) :
    for i in range(0, n) :
        print(arr[i], end = " ")
    print("")


# Driver program to test above function.
A1 = [2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8]
A2 = [2, 1, 8, 3]
m = len(A1)
n = len(A2)
print("Sorted array is ")
sortAccording(A1, A2, m, n)
printArray(A1, m)


# This code is contributed by Nikita Tiwari.
```

C#

```
 // A C# program to sort an array according
// to the order defined by another array
using System;

class GFG {

    /* A Binary Search based function to find
```

```
index of FIRST occurrence of x in arr[].
If x is not present, then it returns -1 */
static int first(int []arr, int low,
                 int high, int x, int n)
{
    if (high >= low)
    {
        /* (low + high)/2; */
        int mid = low + (high - low) / 2;

        if ((mid == 0 || x > arr[mid-1]) &&
                            arr[mid] == x)
            return mid;
        if (x > arr[mid])
            return first(arr, (mid + 1), high,
                                        x, n);
        return first(arr, low, (mid -1), x, n);
    }
    return -1;
}

// Sort A1[0..m-1] according to the order
// defined by A2[0..n-1].
static void sortAccording(int []A1, int []A2,
                                int m, int n)
{

    // The temp array is used to store a copy
    // of A1[] and visited[] is used to mark
    // the visited elements in temp[].
    int []temp = new int[m];
    int []visited = new int[m];

    for (int i = 0; i < m; i++)
    {
        temp[i] = A1[i];
        visited[i] = 0;
    }

    // Sort elements in temp
    Array.Sort(temp);

    // for index of output which is
    // sorted A1[]
    int ind = 0;

    // Consider all elements of A2[], find
    // them in temp[] and copy to A1[] in
```

```
        // order.
        for (int i = 0; i < n; i++)
        {

            // Find index of the first occurrence
            // of A2[i] in temp
            int f = first(temp, 0, m-1, A2[i], m);

            // If not present, no need to proceed
            if (f == -1) continue;

            // Copy all occurrences of A2[i] to A1[]
            for (int j = f; (j < m &&
                           temp[j] == A2[i]); j++)
            {
                A1[ind++] = temp[j];
                visited[j] = 1;
            }
        }

        // Now copy all items of temp[] which are
        // not present in A2[]
        for (int i = 0; i < m; i++)
            if (visited[i] == 0)
                A1[ind++] = temp[i];
    }

    // Utility function to print an array
    static void printArray(int []arr, int n)
    {
        for (int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");
        Console.WriteLine();
    }

    // Driver program to test above function.
    public static void Main()
    {
        int []A1 = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8};
        int []A2 = {2, 1, 8, 3};
        int m = A1.Length;
        int n = A2.Length;
        Console.WriteLine("Sorted array is ");
        sortAccording(A1, A2, m, n);
        printArray(A1, m);
    }
}
```

```
// This code is contributed by nitin mittal.
```

Output:

```
Sorted array is
2 2 1 1 8 8 3 5 6 7 9
```

Method 2 (Using Self-Balancing Binary Search Tree)
We can also use a self balancing BST like AVL Tree, Red Black Tree, etc. Following are detailed steps.
1) Create a self balancing BST of all elements in A1[]. In every node of BST, also keep track of count of occurrences of the key and a bool field visited which is initialized as false for all nodes.
2) Initialize the output index ind as 0.
3) Do following for every element of A2[i] in A2[]
.....a) Search for A2[i] in the BST, if present then copy all occurrences to A1[ind] and increment ind. Also mark the copied elements visited in the BST node.
4) Do an inorder traversal of BST and copy all unvisited keys to A1[].

Time Complexity of this method is same as the previous method. Note that in a self balancing Binary Search Tree, all operations require logm time.

Method 3 (Use Hashing)
1. Loop through A1[], store the count of every number in a HashMap (key: number, value: count of number) .
2. Loop through A2[], check if it is present in HashMap, if so, put in output array that many times and remove the number from HashMap.
3. Sort the rest of the numbers present in HashMap and put in output array.

Thanks toAnurag Sigh for suggesting this method.

The steps 1 and 2 on average take O(m+n) time under the assumption that we have a good hashing function that takes O(1) time for insertion and search on average. The third step takes O(pLogp) time where p is the number of elements remained after considering elements of A2[].

Method 4 (By Writing a Customized Compare Method)
We can also customize compare method of a sorting algorithm to solve the above problem. For example qsort() in C allows us to pass our own customized compare method.
1. If num1 and num2 both are in A2 then number with lower index in A2 will be treated smaller than other.
2. If only one of num1 or num2 present in A2, then that number will be treated smaller than the other which doesn't present in A2.
3. If both are not in A2, then natural ordering will be taken.

Time complexity of this method is O(mnLogm) if we use a O(nLogn) time complexity sorting algorithm. We can improve time complexity to O(mLogm) by using a Hashing instead of doing linear search.

Following is C implementation of this method.

```
 // A C++ program to sort an array according to the order defined
// by another array
#include <stdio.h>
#include <stdlib.h>

// A2 is made global here so that it can be accesed by compareByA2()
// The syntax of qsort() allows only two parameters to compareByA2()
int A2[5];
int size = 5; // size of A2[]

int search(int key)
{
    int i=0, idx = 0;
    for (i=0; i<size; i++)
        if (A2[i] == key)
            return i;
    return -1;
}

// A custom comapre method to compare elements of A1[] according
// to the order defined by A2[].
int compareByA2(const void * a, const void * b)
{
    int idx1 = search(*(int*)a);
    int idx2 = search(*(int*)b);
    if (idx1 != -1 && idx2 != -1)
        return idx1 - idx2;
    else if(idx1 != -1)
        return -1;
    else if(idx2 != -1)
        return 1;
    else
        return ( *(int*)a - *(int*)b );
}

// This method mainly uses qsort to sort A1[] according to A2[]
void sortA1ByA2(int A1[], int size1)
{
    qsort(A1, size1, sizeof (int), compareByA2);
}

// Driver program to test above function
int main(int argc, char *argv[])
{
    int A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8, 7, 5, 6, 9, 7, 5};

       //A2[] = {2, 1, 8, 3, 4};
    A2[0] = 2;
```

```
    A2[1] = 1;
    A2[2] = 8;
    A2[3] = 3;
    A2[4] = 4;
    int size1 = sizeof(A1)/sizeof(A1[0]);

    sortA1ByA2(A1, size1);

    printf("Sorted Array is ");
    int i;
    for (i=0; i<size1; i++)
        printf("%d ", A1[i]);
    return 0;
}
```

Output:

```
Sorted Array is 2 2 1 1 8 8 3 5 5 5 6 6 7 7 7 9 9
```

This method is based on comments by readers (Xinuo Chen, Pranay Doshi and javakurious) and compiled by Anurag Singh.

This article is compiled by **Piyush**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : nitin mittal, Suryaveer Singh

Source

https://www.geeksforgeeks.org/sort-array-according-order-defined-another-array/

Chapter 244

Sort the given string using character search

Sort the given string using character search - GeeksforGeeks

Given a string **str** of size **n**. The problem is to sort the given string without using any sorting techniques (like bubble, selection, etc). The string contains only lowercase characters.

Examples:

```
Input : geeksforgeeks
Output : eeeefggkkorss

Input : coding
Output : cdgino
```

Algorithm:

```
sortString(str, n)
    Initialize new_str = ""

    for i = 'a' to 'z'
        for j = 0 to n-1
            if str[j] == i, then
                new_str += str[j]

    return new_str
```

C++

```
 // C++ implementation to sort the given string without
// using any sorting technique
#include <bits/stdc++.h>
using namespace std;

// function to sort the given string without
// using any sorting technique
string sortString(string str, int n) {

  // to store the final sorted string
  string new_str = "";

  // for each character 'i'
  for (int i = 'a'; i <= 'z'; i++)

    // if character 'i' is present at a particular
    // index then add character 'i' to 'new_str'
    for (int j = 0; j < n; j++)
      if (str[j] == i)
        new_str += str[j];

  // required final sorted string
  return new_str;
}

// Driver program to test above
int main() {
  string str = "geeksforgeeks";
  int n = str.size();
  cout << sortString(str, n);
  return 0;
}
```

Java

```
 // Java implementation to sort the given
// string without using any sorting technique
class GFG {

    // function to sort the given string
    // without using any sorting technique
    static String sortString(String str, int n)
    {

        // to store the final sorted string
        String new_str = "";

        // for each character 'i'
```

```
        for (int i = 'a'; i <= 'z'; i++)

            // if character 'i' is present at a
            // particular index then add character
            // 'i' to 'new_str'
            for (int j = 0; j < n; j++)
                if (str.charAt(j) == i)
                    new_str += str.charAt(j);

        // required final sorted string
        return new_str;
    }

    // Driver code
    public static void main(String[] args)
    {
        String str = "geeksforgeeks";
        int n = str.length();

        System.out.print(sortString(str, n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
 # Python3 implementation to sort
# the given string without using
# any sorting technique

# Function to sort the given string
# without using any sorting technique
def sortString(str, n):

    # To store the final sorted string
    new_str = ""

    # for each character 'i'
    for i in range(ord('a'), ord('z') + 1):

        # if character 'i' is present at a particular
        # index then add character 'i' to 'new_str'
        for j in range(n):
            if (str[j] == chr(i)):
                new_str += str[j]

    # required final sorted string
```

```
    return new_str

# Driver Code
str = "geeksforgeeks"
n = len(str)
print(sortString(str, n))

# This code is contributed by Anant Agarwal.
```

C#

```
 // C# implementation to sort the given
// string without using any sorting technique
using System;

class GFG {

    // function to sort the given string
    // without using any sorting technique
    static String sortString(String str, int n)
    {
        // to store the final sorted string
        String new_str = "";

        // for each character 'i'
        for (int i = 'a'; i <= 'z'; i++)

            // if character 'i' is present at a
            // particular index then add character
            // 'i' to 'new_str'
            for (int j = 0; j < n; j++)
                if (str[j] == i)
                    new_str += str[j];

        // required final sorted string
        return new_str;
    }

    // Driver code
    public static void Main()
    {
        String str = "geeksforgeeks";
        int n = str.Length;

        Console.Write(sortString(str, n));
    }
}
```

```
// This code is contributed by Sam007
```

Output :

```
eeeefggkkorss
```

Source

https://www.geeksforgeeks.org/sort-given-string-using-character-search/

Chapter 245

Squareroot(n)-th node in a Linked List

Squareroot(n)-th node in a Linked List - GeeksforGeeks

Given a Linked List, write a function that accepts the head node of the linked list as a parameter and returns the value of node present at (floor(sqrt(n)))th position in the Linked List, where n is the length of the linked list or the total number of nodes in the list.
Examples:

```
Input : 1->2->3->4->5->NULL
Output : 2

Input : 10->20->30->40->NULL
Output : 20

Input : 10->20->30->40->50->60->70->80->90->NULL
Output : 30
```

Simple method: The simple method is to first find the total number of nodes present in the linked list, then find the value of floor(squareroot(n)) where n is the total number of nodes. Then traverse from the first node in the list to this position and return the node at this position.
This method traverses the linked list 2 times.

Optimized approach: In this method, we can get the required node by traversing the linked list once only. Below is the step by step algorithm for this approach.

1. Initialize two counters i and j both to 1 and a pointer sqrtn to NULL to traverse til the required position is reached.
2. Start traversing the list using head node until the last node is reached.

3. While traversing check if the value of j is equal to sqrt(i). If the value is equal increment both i and j and sqrtn to point sqrtn->next otherwise increment only i.
4. Now, when we will reach the last node of list i will contain value of n, j will contain value of sqrt(i) and sqrtn will point to node at jth position.

```
 // C program to find sqrt(n)'th node
// of a linked list

#include<stdio.h>
#include<stdlib.h>

// Linked list node
struct Node
{
    int data;
    struct Node* next;
};

// Function to get the sqrt(n)th
// node of a linked list
int printsqrtn(struct Node* head)
{
    struct Node* sqrtn = NULL;
    int i = 1, j = 1;

    // Traverse the list
    while (head!=NULL)
    {
        // check if j = sqrt(i)
        if (i == j*j)
        {
            // for first node
            if (sqrtn == NULL)
                sqrtn = head;
            else
                sqrtn = sqrtn->next;

            // increment j if j = sqrt(i)
            j++;
        }
        i++;

        head=head->next;
    }

    // return node's data
    return sqrtn->data;
}
```

```
void print(struct Node* head)
{
    while (head != NULL)
    {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

// function to add a new node at the
// begining of the list
void push(struct Node** head_ref, int new_data)
{
    // allocate node
    struct Node* new_node =
            (struct Node*) malloc(sizeof(struct Node));

    // put in the data
    new_node->data = new_data;

    // link the old list off the new node
    new_node->next = (*head_ref);

    // move the head to point to the new node
    (*head_ref) = new_node;
}

/* Driver program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;
    push(&head, 40);
    push(&head, 30);
    push(&head, 20);
    push(&head, 10);
    printf("Given linked list is:");
    print(head);
    printf("sqrt(n)th node is %d ",printsqrtn(head));

    return 0;
}
```

Output:

```
Given linked list is:10 20 30 40
sqrt(n)th node is 20
```

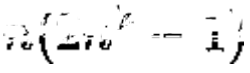
Source

https://www.geeksforgeeks.org/squarerootnth-node-in-a-linked-list/

Chapter 246

Stella Octangula Number

Stella Octangula Number - GeeksforGeeks

Given a number n, check it is the **Stella Octangula number** or not. A number of the form  where n is a whole number(0, 1, 2, 3, 4, ...) is called Stella Octangula. First few Stella Octangula numbers are 0, 1, 14, 51, 124, 245, 426, 679, 1016, 1449, 1990, ...

Stella octangula numbers which are perfect squares are **1** and **9653449**.

Given a number x, check if it is Stella octangula.

Examples:

```
Input: x = 51
Output: Yes
For n = 3, the value of expression
n(2n2 - 1) is 51

Input: n = 53
Output: No
```

A **simple solution** is to run a loop starting from n = 0. For every n, check if $n(2n^2 - 1)$ is equal to x. We run the loop while value of $n(2n^2 - 1)$ is smaller than or equal to x.

An **efficient solution** is to use Unbounded Binary Search. We first find a value of n such that $n(2n^2 - 1)$ is greater than x using repeated doubling. Then we apply Binary Search.

C++

```
 // Program to check if a number is Stella
// Octangula Number
#include <bits/stdc++.h>
using namespace std;
```

```
// Returns value of n*(2*n*n - 1)
int f(int n) {
   return n*(2*n*n - 1);
}

// Finds if a value of f(n) is equl to x
// where n is in interval [low..high]
bool binarySearch(int low, int high, int x)
{
    while (low <= high) {
        long long mid = (low + high) / 2;

        if (f(mid) < x)
            low = mid + 1;
        else if (f(mid) > x)
            high = mid - 1;
        else
            return true;
    }
    return false;
}

// Returns true if x isStella Octangula Number.
// Else returns false.
bool isStellaOctangula(int x)
{
    if (x == 0)
      return true;

    // Find 'high' for binary search by
    // repeated doubling
    int i = 1;
    while (f(i) < x)
        i = i*2;

    // If condition is satisfied for a
    // power of 2.
    if (f(i) == x)
        return true;

    //  Call binary search
    return binarySearch(i/2, i, x);
}

// driver code
int main()
{
```

```
    int n = 51;
    if (isStellaOctangula(n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
 // Program to check if
// a number is Stella
// Octangula Number
import java.io.*;

class GFG
{

// Returns value of
// n*(2*n*n - 1)
static int f(int n)
{
    return n * (2 * n *
                n - 1);
}

// Finds if a value of
// f(n) is equl to x
// where n is in
// interval [low..high]
static boolean binarySearch(int low,
                            int high,
                            int x)
{
    while (low <= high)
    {
        int mid = (low + high) / 2;

        if (f(mid) < x)
            low = mid + 1;
        else if (f(mid) > x)
            high = mid - 1;
        else
            return true;
    }
    return false;
}
```

```
// Returns true if x
// is Stella Octangula
// Number.Else returns
// false.
static boolean isStellaOctangula(int x)
{
    if (x == 0)
    return true;

    // Find 'high' for
    // binary search by
    // repeated doubling
    int i = 1;
    while (f(i) < x)
        i = i * 2;

    // If condition is
    // satisfied for a
    // power of 2.
    if (f(i) == x)
        return true;

    // Call binary search
    return binarySearch(i / 2,
                        i, x);
}

// Driver code
public static void main (String[] args)
{
    int n = 51;
    if (isStellaOctangula(n))
        System.out.print("Yes");
    else
        System.out.print("No");
}
}

// This code is contributed
// by anuj_67.
```

C#

```
 // Program to check if
// a number is Stella
// Octangula Number
using System;
```

```
class GFG
{

// Returns value of
// n*(2*n*n - 1)
static int f(int n)
{
    return n * (2 * n *
                n - 1);
}

// Finds if a value of
// f(n) is equl to x
// where n is in
// interval [low..high]
static bool binarySearch(int low,
                         int high,
                         int x)
{
    while (low <= high)
    {
        int mid = (low + high) / 2;

        if (f(mid) < x)
            low = mid + 1;
        else if (f(mid) > x)
            high = mid - 1;
        else
            return true;
    }
    return false;
}

// Returns true if x
// is Stella Octangula
// Number.Else returns
// false.
static bool isStellaOctangula(int x)
{
    if (x == 0)
    return true;

    // Find 'high' for
    // binary search by
    // repeated doubling
    int i = 1;
    while (f(i) < x)
```

```
        i = i * 2;

    // If condition is
    // satisfied for a
    // power of 2.
    if (f(i) == x)
        return true;

    // Call binary search
    return binarySearch(i / 2,
                        i, x);
}

// Driver code
public static void Main ()
{
    int n = 51;
    if (isStellaOctangula(n))
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
}
}

// This code is contributed
// by anuj_67.
```

Output:

```
Yes
```

Improved By : vt_m

Source

https://www.geeksforgeeks.org/stella-octangula-number/

Chapter 247

Structure Sorting (By Multiple Rules) in C++

Structure Sorting (By Multiple Rules) in C++ - GeeksforGeeks

Prerequisite : Structures in C

Name and marks in different subjects (physics, chemistry and maths) are given for all students. The task is to compute total marks and ranks of all students. And finally display all students sorted by rank.

Rank of student is computed using below rules.

1. If total marks are different, then students with higher marks gets better rank.
2. If total marks are same, then students with higher marks in Maths gets better rank.
3. If total marks are same and marks in Maths are also same, then students with better marks in Physics gets better rank.
4. If all marks (total, Maths, Physics and Chemistry) are same, then any student can be assigned bedtter rank.

We use below structure to store details of students.

```
struct Student
{
    string name; // Given
    int math;  // Marks in math (Given)
    int phy;   // Marks in Physics (Given)
    int che;   // Marks in Chemistry (Given)
    int total; // Total marks (To be filled)
    int rank;  // Rank of student (To be filled)
};
```

We use std::sort() for **Structure Sorting**. In Structure sorting, all the respective properties possessed by the structure object are sorted on the basis of one (or more) property of the object.

In this example, marks of students in different subjects are provided by user. These marks in individual subjects are added to calculate the total marks of the student, which is then used to sort different students on the basis of their ranks (as explained above).

```
 // C++ program to demonstrate structure sorting in C++
#include <bits/stdc++.h>
using namespace std;

struct Student
{
    string name; // Given
    int math;  // Marks in math (Given)
    int phy;   // Marks in Physics (Given)
    int che;   // Marks in Chemistry (Given)
    int total; // Total marks (To be filled)
    int rank;  // Rank of student (To be filled)
};

// Function for comparing two students according
// to given rules
bool compareTwoStudents(Student a, Student b)
{
    // If total marks are not same then
    // returns true for higher total
    if (a.total != b.total )
        return a.total > b.total;

    // If marks in Maths are not same then
    // returns true for higher marks
    if (a.math != b.math)
            return a.math > b.math;

    return (a.phy > b.phy);
}

// Fills total marks and ranks of all Students
void computeRanks(Student a[], int n)
{
    // To calculate total marks for all Students
    for (int i=0; i<n; i++)
        a[i].total = a[i].math + a[i].phy + a[i].che;

    // Sort structure array using user defined
    // function compareTwoStudents()
    sort(a, a+5, compareTwoStudents);
```

```
    // Assigning ranks after sorting
    for (int i=0; i<n; i++)
        a[i].rank = i+1;
}

// Driver code
int main()
{
    int n = 5;

    // array of structure objects
    Student a[n];

    // Details of Student 1
    a[0].name = "Bryan" ;
    a[0].math = 80 ;
    a[0].phy = 95 ;
    a[0].che = 85 ;

    // Details of Student 2
    a[1].name= "Kevin" ;
    a[1].math= 95 ;
    a[1].phy= 85 ;
    a[1].che= 99 ;

    // Details of Student 3
    a[2].name = "Nick" ;
    a[2].math = 95 ;
    a[2].phy = 85 ;
    a[2].che = 80 ;

    // Details of Student 4
    a[3].name = "AJ" ;
    a[3].math = 80 ;
    a[3].phy = 70 ;
    a[3].che = 90 ;

    // Details of Student 5
    a[4].name = "Howie" ;
    a[4].math = 80 ;
    a[4].phy = 80 ;
    a[4].che = 80 ;

    computeRanks(a, n);

    //Column names for displaying data
    cout << "Rank" <<"t" << "Name" << "t";
```

```
    cout << "Maths" <<"t" <<"Physics" <<"t"
         << "Chemistry";
    cout << "t" << "Totaln";

    // Display details of Students
    for (int i=0; i<n; i++)
    {
        cout << a[i].rank << "t";
        cout << a[i].name << "t";
        cout << a[i].math << "t"
             << a[i].phy << "t"
             << a[i].che << "tt";
        cout << a[i].total <<"t";
        cout <<"n";
    }

    return 0;
}
```

Output:

```
Rank    Name    Maths   Physics Chemistry   Total
1   Kevin   95  85  99      279
2   Nick    95  85  80      260
3   Bryan   80  95  85      260
4   Howie   80  80  80      240
5   AJ  80  70  90      240
```

Related Articles:
sort() in C++ STL
Comparator function of qsort() in C
C qsort() vs C++ sort()
Sort an array according to count of set bits

Source

https://www.geeksforgeeks.org/structure-sorting-in-c/

Chapter 248

Sublist Search (Search a linked list in another list)

Sublist Search (Search a linked list in another list) - GeeksforGeeks

Given two linked lists, the task is to check whether the first list is present in 2nd list or not.

```
Input  : list1 =  10->20
         list2  = 5->10->20
Output : LIST FOUND

Input  : list1 =  1->2->3->4
         list2  = 1->2->1->2->3->4
Output : LIST FOUND

Input  : list1 =  1->2->3->4
         list2  = 1->2->2->1->2->3
Output : LIST NOT FOUND
```

Algorithm:
1- Take first node of second list.
2- Start matching the first list from this first node.
3- If whole lists match return true.
4- Else break and take first list to the first node again.
5- And take second list to its second node.
6- Repeat these steps until any of linked lists becomes empty.
7- If first list becomes empty then list found else not.

Below is C++ implementation.

```
// C++ program to find a list in second list
```

```
#include <bits/stdc++.h>
using namespace std;

// A Linked List node
struct Node
{
    int data;
    Node* next;
};

// Returns true if first list is present in second
// list
bool findList(Node* first, Node* second)
{
    Node* ptr1 = first, *ptr2 = second;

    // If both linked lists are empty, return true
    if (first == NULL && second == NULL)
        return true;

    // Else If one is empty and other is not return
    // false
    if ( first == NULL ||
        (first != NULL && second == NULL))
        return false;

    // Traverse the second list by picking nodes
    // one by one
    while (second != NULL)
    {
        // Initialize ptr2 with current node of second
        ptr2 = second;

        // Start matching first list with second list
        while (ptr1 != NULL)
        {
            // If second list becomes empty and first
            // not then return false
            if (ptr2 == NULL)
                return false;

            // If data part is same, go to next
            // of both lists
            else if (ptr1->data == ptr2->data)
            {
                ptr1 = ptr1->next;
                ptr2 = ptr2->next;
            }
```

```
            // If not equal then  break the loop
            else break;
        }

        // Return true if first list gets traversed
        // completely that means it is matched.
        if (ptr1 == NULL)
            return true;

        // Initialize ptr1 with first again
        ptr1 = first;

        // And go to next node of second list
        second = second->next;
    }

    return false;
}

/* Function to print nodes in a given linked list */
void printList(Node* node)
{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

// Function to add new node to linked lists
Node *newNode(int key)
{
    Node *temp = new Node;
    temp-> data= key;
    temp->next = NULL;
    return temp;
}

/* Driver program to test above functions*/
int main()
{
    /* Let us create two linked lists to test
    the above functions. Created lists shall be
        a: 1->2->3->4
        b: 1->2->1->2->3->4*/
    Node *a = newNode(1);
    a->next = newNode(2);
```

```
    a->next->next = newNode(3);
    a->next->next->next = newNode(4);

    Node *b = newNode(1);
    b->next = newNode(2);
    b->next->next = newNode(1);
    b->next->next->next = newNode(2);
    b->next->next->next->next = newNode(3);
    b->next->next->next->next->next = newNode(4);

    findList(a,b) ? cout << "LIST FOUND" :
                    cout << "LIST NOT FOUND";

    return 0;
}
```

Output:

```
LIST FOUND
```

Time Complexity : O(m*n) where m is the number of nodes in second list and n in first.

Optimization :
Above code can be optimized by using extra space i.e. stores the list into two strings and apply KMP algorithm. Refer https://ide.geeksforgeeks.org/3fXUaV for implementation provided by **Nishant Singh**.

Source

https://www.geeksforgeeks.org/sublist-search-search-a-linked-list-in-another-list/

Chapter 249

Sudo Placement | Beautiful Pairs

Sudo Placement | Beautiful Pairs - GeeksforGeeks

Given two arrays of integers where maximum size of first array is big and that of second array is small. Your task is to find if there is a pair in the first array whose sum is present in the second array.

Examples:

```
Input:
4
1 5 10 8
3
2 20 13
Output: 1
```

Approach : We have x + y = z. This can be rewritten as x = z – y. This means, we need to find an element x in array 1 such that it is the result of z(second array) – y(first array). For this, use hashing to keep track of such element x.

```
 // CPP code for finding required pairs
#include <bits/stdc++.h>
using namespace std;

// The function to check if beautiful pair exists
bool pairExists(int arr1[], int m, int arr2[], int n)
{
    // Set for hashing
    unordered_set<int> s;
```

```
    // Traversing the first array
    for (int i = 0; i < m; i++) {

        // Traversing the second array to check for
        // every j corresponding to single i
        for (int j = 0; j < n; j++) {

            // x + y = z => x = y - z
            if (s.find(arr2[j] - arr1[i]) != s.end())

                // if such x exists then we return true
                return true;

        }

       // hash to make use of it next time
       s.insert(arr1[i]);
 }

    // no pair exists
    return false;
}

// Driver Code
int main()
{
    int arr1[] = { 1, 5, 10, 8 };
    int arr2[] = { 2, 20, 13 };

    // If pair exists then 1 else 0
    // 2nd argument as size of first array
    // fourth argument as sizeof 2nd array
    if (pairExists(arr1, 4, arr2, 3))
        cout << 1 << endl;
    else
        cout << 0 << endl;

    return 0;
}
```

Output:

```
1
```

Source

https://www.geeksforgeeks.org/sudo-placement-beautiful-pairs/

Chapter 250

Sudo Placement | Placement Tour

Sudo Placement | Placement Tour - GeeksforGeeks

Given an array A of N positive integers and a budget B. Your task is to decide the maximum number of elements to be picked from the array such that the cumulative cost of all picked elements is less than or equal to budget B. Cost of picking the ith element is given by : A[i] + (i * K) where, K is a constant whose value is equal to the number of elements picked. The indexing(i) is 1 based. Print the maximum number and its respective cumulative cost.

Examples:

> **Input** : arr[] = { 2, 3, 5 }, B = 11
> **Output** : 2 11
> **Explanation** : Cost of picking maximum elements = {2 + (1 * 2) } + {3 + (2 * 2)} = 4 + 7 = 11 (which is equal to budget)
>
> **Input** : arr[] = { 1, 2, 5, 6, 3 }, B = 90
> **Output** : 4 54

Prerequisites : Binary Search

Approach: The idea here is to use binary search on all possible values of K i.e. the optimal number of elements to be picked. *Start* with zero as lower bound and *End* with total number of elements i.e. N as upper bound. Check if by setting K as current *Mid*, obtained cumulative cost is less than or equal to budget. If it satisfies the condition, then try to increase K by setting *Start* as *(Mid + 1)*, otherwise try to decrease K by setting *End* as *(Mid – 1)*.
Checking of the condition can be done in a brute force manner by simply modifying the array according to the given formula and adding the K (current number of elements to be picked) smallest modified values to get the cumulative cost.

Below is the implementation of above approach.

```
 // CPP Program to find the optimal number of
// elements such that the cumulative value
// should be less than given number
#include <bits/stdc++.h>

using namespace std;

// This function returns true if the value cumulative
// according to received integer K is less than budget
// B, otherwise returns false
bool canBeOptimalValue(int K, int arr[], int N, int B,
                       int& value)
{
    // Initialize a temporary array which stores
    // the cumulative value of the original array
    int tmp[N];

    for (int i = 0; i < N; i++)
        tmp[i] = (arr[i] + K * (i + 1));

    // Sort the array to find the smallest K values
    sort(tmp, tmp + N);

    value = 0;
    for (int i = 0; i < K; i++)
        value += tmp[i];

    // Check if the value is less than budget
    return value <= B;
}

// This function prints the optimal number of elements
// and respective cumulative value which is less than
// the given number
void findNoOfElementsandValue(int arr[], int N, int B)
{
    int start = 0; // Min Value or lower bound

    int end = N; // Max Value or upper bound

    // Initialize answer as zero as optimal value
    // may not exists
    int ans = 0;

    int cumulativeValue = 0;

    while (start <= end) {
        int mid = (start + end) / 2;
```

```
        // If the current Mid Value is an optimal
        // value, then try to maximize it
        if (canBeOptimalValue(mid, arr, N, B,
                              cumulativeValue)) {
            ans = mid;
            start = mid + 1;
        }
        else
            end = mid - 1;
    }
    // Call Again to set the corresponding cumulative
    // value for the optimal ans
    canBeOptimalValue(ans, arr, N, B, cumulativeValue);

    cout << ans << " " << cumulativeValue << endl;
}

// Driver Code
int main()
{
    int arr[] = { 1, 2, 5, 6, 3 };
    int N = sizeof(arr) / sizeof(arr[0]);

    // Budget
    int B = 90;
    findNoOfElementsandValue(arr, N, B);
    return 0;
}
```

Output:

```
4 54
```

Time Complexity: O(N * $(\log N)^2$), where N is the number of elements in the given array.

Source

https://www.geeksforgeeks.org/sudo-placement-placement-tour/

Chapter 251

Super Prime

Super Prime - GeeksforGeeks

Super-prime numbers (also known as **higher order primes**) are the subsequence of prime numbers that occupy prime-numbered positions within the sequence of all prime numbers. First few Super-Primes are 3, 5, 11 and 17.

The task is to print all the Super-Primes less than or equal to the given positive integer N.

Examples:

```
Input: 7
Output: 3 5
3 is super prime because it appears at second
position in list of primes (2, 3, 5, 7, 11, 13,
17, 19, 23, ...) and 2 is also prime. Similarly
5 appears at third position and 3 is a prime.

Input: 17
Output: 3 5 11 17
```

The idea is to generate all the primes less than or equal to the given number N using Sieve of Eratosthenes. Once we have generated all such primes, we iterate through all numbers and store it in the array. Once we have stored all the primes in the array, we iterate through the array and print all prime number which occupies prime number position in the array.

C/C++

```
 // C++ program to print super primes less than
// or equal to n.
#include<bits/stdc++.h>
using namespace std;
```

```
// Generate all prime numbers less than n.
bool SieveOfEratosthenes(int n, bool isPrime[])
{
    // Initialize all entries of boolean array
    // as true. A value in isPrime[i] will finally
    // be false if i is Not a prime, else true
    // bool isPrime[n+1];
    isPrime[0] = isPrime[1] = false;
    for (int i=2; i<=n; i++)
        isPrime[i] = true;

    for (int p=2; p*p<=n; p++)
    {
        // If isPrime[p] is not changed, then it is
        // a prime
        if (isPrime[p] == true)
        {
            // Update all multiples of p
            for (int i=p*2; i<=n; i += p)
                isPrime[i] = false;
        }
    }
}

// Primts all super primes less than or equal to n.
void superPrimes(int n)
{
    // Generating primes using Sieve
    bool isPrime[n+1];
    SieveOfEratosthenes(n, isPrime);

    // Storing all the primes generated in a
    // an array primes[]
    int primes[n+1], j = 0;
    for (int p=2; p<=n; p++)
       if (isPrime[p])
           primes[j++] = p;

    // Printing all those prime numbers that
    // occupy prime numbered position in
    // sequence of prime numbers.
    for (int k=0; k<j; k++)
        if (isPrime[k+1])
            cout << primes[k] << " ";
}

// Driven program
int main()
```

```
{
    int n = 241;
    cout << "Super-Primes less than or equal to "
         << n << " are :"<<endl;
    superPrimes(n);
    return 0;
}
```

Java

```
 // Java program to print super
// primes less than or equal to n.
import java.io.*;

class GFG {

    // Generate all prime
    // numbers less than n.
    static void SieveOfEratosthenes
                (int n, boolean isPrime[])
    {
        // Initialize all entries of boolean
        // array as true. A value in isPrime[i]
        // will finally be false if i is Not
        // a prime, else true bool isPrime[n+1];
        isPrime[0] = isPrime[1] = false;
        for (int i=2; i<=n; i++)
            isPrime[i] = true;

        for (int p=2; p*p<=n; p++)
        {
            // If isPrime[p] is not changed,
            // then it is a prime
            if (isPrime[p] == true)
            {
                // Update all multiples of p
                for (int i=p*2; i<=n; i += p)
                    isPrime[i] = false;
            }
        }
    }

    // Primts all super primes less
    // than or equal to n.
    static void superPrimes(int n)
    {

        // Generating primes using Sieve
```

```
        boolean isPrime[]=new boolean[n+1];
        SieveOfEratosthenes(n, isPrime);

        // Storing all the primes generated
        // in a an array primes[]
        int primes[] = new int[n+1];
        int j = 0;

        for (int p=2; p<=n; p++)
            if (isPrime[p])
                primes[j++] = p;

        // Printing all those prime numbers that
        // occupy prime numbered position in
        // sequence of prime numbers.
        for (int k=0; k<j; k++)
            if (isPrime[k+1])
                System.out.print(primes[k]+ " ");
    }

    // Driven program
    public static void main(String args[])
    {
        int n = 241;
        System.out.println("Super-Primes less than or equal to "
                           +n+" are :");
        superPrimes(n);
    }
}

// This code is contributed by Nikita Tiwari.
```

Python

```
 # Python program to print super primes less than
# or equal to n.

# Generate all prime numbers less than n.
def SieveOfEratosthenes(n, isPrime):
    # Initialize all entries of boolean array
    # as true. A value in isPrime[i] will finally
    # be false if i is Not a prime, else true
    # bool isPrime[n+1]
    isPrime[0] = isPrime[1] = False
    for i in range(2,n+1):
        isPrime[i] = True

    for p in range(2,n+1):
```

```
        # If isPrime[p] is not changed, then it is
        # a prime
        if (p*p<=n and isPrime[p] == True):
            # Update all multiples of p
            for i in range(p*2,n+1,p):
                isPrime[i] = False
                p += 1
def superPrimes(n):

    # Generating primes using Sieve
    isPrime = [1 for i in range(n+1)]
    SieveOfEratosthenes(n, isPrime)

    # Storing all the primes generated in a
    # an array primes[]
    primes = [0 for i in range(2,n+1)]
    j = 0
    for p in range(2,n+1):
       if (isPrime[p]):
           primes[j] = p
           j += 1

    # Printing all those prime numbers that
    # occupy prime numbered position in
    # sequence of prime numbers.
    for k in range(j):
        if (isPrime[k+1]):
            print primes[k],

n = 241
print "Super-Primes less than or equal to ", n, " are :n",
superPrimes(n)

# Contributed by: Afzal
```

C#

```
 // Program to print super primes
// less than or equal to n.
using System;

class GFG {

    // Generate all prime
    // numbers less than n.
    static void SieveOfEratosthenes(int n, bool[] isPrime)
    {
        // Initialize all entries of boolean
```

```
    // array as true. A value in isPrime[i]
    // will finally be false if i is Not
    // a prime, else true bool isPrime[n+1];
    isPrime[0] = isPrime[1] = false;

    for (int i = 2; i <= n; i++)
        isPrime[i] = true;

    for (int p = 2; p * p <= n; p++) {
        // If isPrime[p] is not changed,
        // then it is a prime
        if (isPrime[p] == true) {
            // Update all multiples of p
            for (int i = p * 2; i <= n; i += p)
                isPrime[i] = false;
        }
    }
}

// Prints all super primes less
// than or equal to n.
static void superPrimes(int n)
{

    // Generating primes using Sieve
    bool[] isPrime = new bool[n + 1];
    SieveOfEratosthenes(n, isPrime);

    // Storing all the primes generated
    // in a an array primes[]
    int[] primes = new int[n + 1];
    int j = 0;

    for (int p = 2; p <= n; p++)
        if (isPrime[p])
            primes[j++] = p;

    // Printing all those prime numbers
    // that occupy prime number position
    // in sequence of prime numbers.
    for (int k = 0; k < j; k++)
        if (isPrime[k + 1])
            Console.Write(primes[k] + " ");
}

// Driven program
public static void Main()
{
```

```
        int n = 241;
        Console.WriteLine("Super-Primes less than or equal to "
                          + n + " are :");
        superPrimes(n);
    }
}

// This code is contributed by Anant Agarwal.
```

Output:

```
Super-Primes less than or equal to 241 are :
3 5 11 17 31 41 59 67 83 109 127 157 179 191 211 241
```

References: https://en.wikipedia.org/wiki/Super-prime

Source

https://www.geeksforgeeks.org/super-prime/

Chapter 252

The Ubiquitous Binary Search | Set 1

The Ubiquitous Binary Search | Set 1 - GeeksforGeeks

We all aware of binary search algorithm. Binary search is easiest difficult algorithm to get it right. I present some interesting problems that I collected on binary search. There were some requests on binary search.

I request you to honor the code, "I sincerely attempt to solve the problem and ensure there are no corner cases". After reading each problem minimize the browser and try solving it.

Problem Statement: Given a sorted array of N distinct elements. Find a key in the array using least number of comparisons. (Do you think binary search is optimal to search a key in sorted array?)

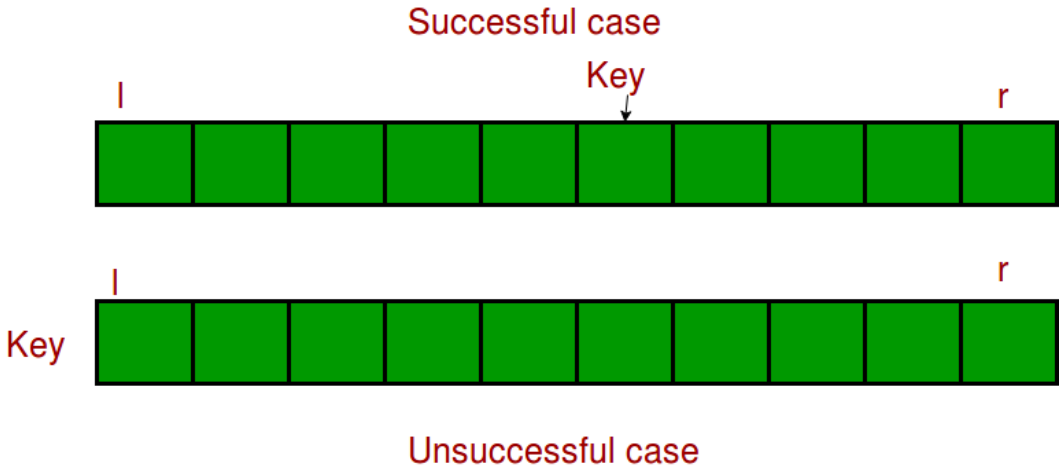
Without much theory, here is typical binary search algorithm.

```
 // Returns location of key, or -1 if not found
int BinarySearch(int A[], int l, int r, int key)
{
    int m;

    while( l <= r )
    {
        m = l + (r-l)/2;

        if( A[m] == key ) // first comparison
            return m;

        if( A[m] < key ) // second comparison
            l = m + 1;
        else
```

```
            r = m - 1;
    }

    return -1;
}
```

Theoretically we need *log N + 1* comparisons in worst case. If we observe, we are using two comparisons per iteration except during final successful match, if any. In practice, comparison would be costly operation, it won't be just primitive type comparison. It is more economical to minimize comparisons as that of theoretical limit.

See below figure on initialize of indices in the next implementation.



The following implementation uses fewer number of comparisons.

```
 // Invariant: A[l] <= key and A[r] > key
// Boundary: |r - l| = 1
// Input: A[l .... r-1]
int BinarySearch(int A[], int l, int r, int key)
{
    int m;

    while( r - l > 1 )
    {
        m = l + (r-l)/2;

        if( A[m] <= key )
            l = m;
        else
            r = m;
    }

    if( A[l] == key )
```

```
        return l;
    else
        return -1;
}
```

In the while loop we are depending only on one comparison. The search space converges to place l and r point two different consecutive elements. We need one more comparison to trace search status.

You can see sample test case http://ideone.com/76bad0. (*C++11 code*)

Problem Statement: Given an array of N distinct integers, find floor value of input 'key'. Say, A = {-1, 2, 3, 5, 6, 8, 9, 10} and key = 7, we should return 6 as outcome.

We can use the above optimized implementation to find floor value of key. We keep moving the left pointer to right most as long as the invariant holds. Eventually left pointer points an element less than or equal to key (by definition floor value). The following are possible corner cases,

—> If all elements in the array are smaller than key, left pointer moves till last element.

—> If all elements in the array are greater than key, it is an error condition.

—> If all elements in the array equal and <= key, it is worst case input to our implementation.

Here is implementation,

```
 // largest value <= key
// Invariant: A[l] <= key and A[r] > key
// Boundary: |r - l| = 1
// Input: A[l .... r-1]
// Precondition: A[l] <= key <= A[r]
int Floor(int A[], int l, int r, int key)
{
    int m;

    while( r - l > 1 )
    {
        m = l + (r - l)/2;

        if( A[m] <= key )
            l = m;
        else
            r = m;
    }

    return A[l];
}

// Initial call
```

```
int Floor(int A[], int size, int key)
{
    // Add error checking if key < A[0]
    if( key < A[0] )
        return -1;

    // Observe boundaries
    return Floor(A, 0, size, key);
}
```

You can see some test cases http://ideone.com/z0Kx4a.

Problem Statement: Given a sorted array with possible duplicate elements. Find number of occurrences of input 'key' in *log N* time.

The idea here is finding left and right most occurrences of key in the array using binary search. We can modify floor function to trace right most occurrence and left most occurrence. Here is implementation,

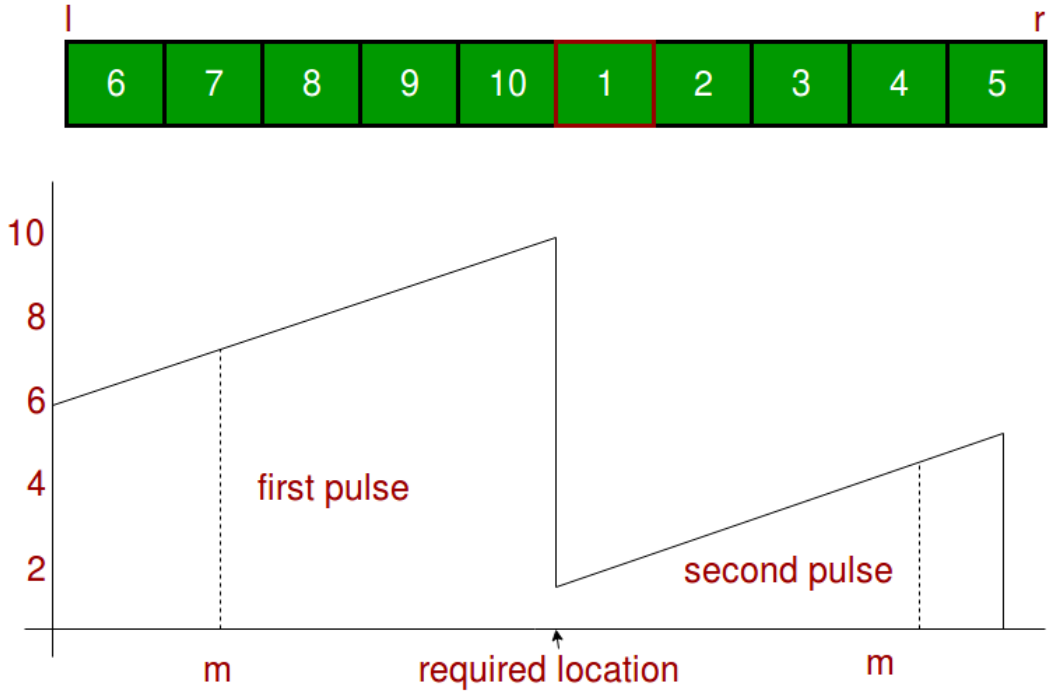
```
 // Input: Indices Range [l ... r)
// Invariant: A[l] <= key and A[r] > key
int GetRightPosition(int A[], int l, int r, int key)
{
    int m;

    while( r - l > 1 )
    {
        m = l + (r - l)/2;

        if( A[m] <= key )
            l = m;
        else
            r = m;
    }

    return l;
}

// Input: Indices Range (l ... r]
// Invariant: A[r] >= key and A[l] > key
int GetLeftPosition(int A[], int l, int r, int key)
{
    int m;

    while( r - l > 1 )
    {
        m = l + (r - l)/2;

        if( A[m] >= key )
```

```
            r = m;
        else
            l = m;
    }

    return r;
}

int CountOccurances(int A[], int size, int key)
{
    // Observe boundary conditions
    int left = GetLeftPosition(A, -1, size-1, key);
    int right = GetRightPosition(A, 0, size, key);

    // What if the element doesn't exists in the array?
    // The checks helps to trace that element exists
    return (A[left] == key && key == A[right])?
           (right - left + 1) : 0;
}
```

Sample code http://ideone.com/zn6R6a.

Problem Statement: Given a sorted array of distinct elements, and the array is rotated at an unknown position. Find minimum element in the array.

We can see pictorial representation of sample input array in the below figure.



Graphical Representation

We converge the search space till l and r points single element. If the middle location falls in the first pulse, the condition A[m] < A[r] doesn't satisfy, we exclude the range A[m+1 … r]. If the middle location falls in the second pulse, the condition A[m] < A[r] satisfied, we converge our search space to A[m+1 … r]. At every iteration we check for search space size, if it is 1, we are done.

Given below is implementation of algorithm. *Can you come up with different implementation?*

```
 int BinarySearchIndexOfMinimumRotatedArray(int A[], int l, int r)
{
    // extreme condition, size zero or size two
    int m;

    // Precondition: A[l] > A[r]
    if( A[l] <= A[r] )
        return l;

    while( l <= r )
    {
        // Termination condition (l will eventually falls on r, and r always
        // point minimum possible value)
        if( l == r )
```

```
            return l;

        m = l + (r-l)/2; // 'm' can fall in first pulse,
                         // second pulse or exactly in the middle

        if( A[m] < A[r] )
            // min can't be in the range
            // (m < i <= r), we can exclude A[m+1 ... r]
            r = m;
        else
            // min must be in the range (m < i <= r),
            // we must search in A[m+1 ... r]
            l = m+1;
    }

    return -1;
}

int BinarySearchIndexOfMinimumRotatedArray(int A[], int size)
{
    return BinarySearchIndexOfMinimumRotatedArray(A, 0, size-1);
}
```

See sample test cases http://ideone.com/KbwDrk.

Exercises:

1. A function called *signum(x, y)* is defined as,

```
signum(x, y) = -1 if x < y
             =  0 if x = y
             =  1 if x > y
```

Did you come across any instruction set in which a comparison behaves like *signum* function? Can it make first implementation of binary search optimal?

2. Implement ceil function replica of floor function.

3. Discuss with your friends on "Is binary search optimal (results in least number of comparisons)? Why not ternary search or interpolation search on sorted array? When do you prefer ternary or interpolation search over binary search?"

4. Draw a tree representation of binary search (believe me, it helps you a lot to understand many internals of binary search).

Stay tuned, I will cover few more interesting problems using binary search in upcoming articles. I welcome your comments.

– – – by **Venki**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

https://www.geeksforgeeks.org/the-ubiquitous-binary-search-set-1/

Chapter 253

The painter's partition problem

The painter's partition problem - GeeksforGeeks

We have to paint n boards of length {A1, A2…An}. There are k painters available and each takes 1 unit time to paint 1 unit of board. The problem is to find the minimum time to get this job done under the constraints that any painter will only paint continuous sections of boards, say board {2, 3, 4} or only board {1} or nothing but not board {2, 4, 5}.

Examples:

```
Input : k = 2, A = {10, 10, 10, 10}
Output : 20.
Here we can divide the boards into 2
equal sized partitions, so each painter
gets 20 units of board and the total
time taken is 20.

Input : k = 2, A = {10, 20, 30, 40}
Output : 60.
Here we can divide first 3 boards for
one painter and the last board for
second painter.
```

From the above examples, it is obvious that the strategy of dividing the boards into k equal partitions won't work for all the cases. We can observe that the problem can be **broken down** into: Given an array A of non-negative integers and a positive integer k, we have to divide A into k of fewer partitions such that the maximum sum of the elements in a partition, overall partitions is minimized. So for the second example above, possible **divisions** are:
* One partition: so time is 100.
* Two partitions: (10) & (20, 30, 40), so time is 90. Similarly we can put the first divider after 20 (=> time 70) or 30 (=> time 60); so this means the minimum time: (100, 90, 70, 60) is 60.

A **brute force** solution is to consider all possible set of contiguous partitions and calculate the maximum sum partition in each case and return the minimum of all these cases.

1) Optimal Substructure:
We can implement the naive solution using recursion with the following optimal substructure property:
Assuming that we already have k-1 partitions in place (using k-2 dividers), we now have to put the k-1 th divider to get k partitions.
How can we do this? We can put the k-1 th divider between the i th and i+1 th element where i = 1 to n. Please note that putting it before the first element is the same as putting it after the last element.

The total cost of this arrangement can be calculated as the **maximum** of the following:
a) The cost of the last partition: sum(Ai..An), where the k-1 th divider is
before element i.
b) The maximum cost of any partition already formed to the left of the k-1 th divider.

Here a) can be found out using a simple **helper function** to calculate sum
of elements between two indices in the array. How to find out b) ?
We can observe that b) actually is to place the k-2 separators as fairly as
possible, so it is a **subproblem** of the given problem. Thus we can write the optimal substructure property as the following recurrence relation:

$$T(n,k) = min\left\{ max_{i=1}^{n}\left\{ T(i,k-1), \sum_{j=i+1}^{n} A_j \right\} \right\}$$

The base case are:

$$T(1,k) = A1$$

$$T(n,1) = \sum_{i=1}^{n} A_i$$

Following is the implementation of the above recursive equation:

C++

```
 // CPP program for The painter's partition problem
#include <climits>
#include <iostream>
using namespace std;

// function to calculate sum between two indices
// in array
int sum(int arr[], int from, int to)
```

```
{
    int total = 0;
    for (int i = from; i <= to; i++)
        total += arr[i];
    return total;
}

// for n boards and k partitions
int partition(int arr[], int n, int k)
{
    // base cases
    if (k == 1) // one partition
        return sum(arr, 0, n - 1);
    if (n == 1)  // one board
        return arr[0];

    int best = INT_MAX;

    // find minimum of all possible maximum
    // k-1 partitions to the left of arr[i],
    // with i elements, put k-1 th divider
    // between arr[i-1] & arr[i] to get k-th
    // partition
    for (int i = 1; i <= n; i++)
        best = min(best, max(partition(arr, i, k - 1),
                             sum(arr, i, n - 1)));

    return best;
}

int main()
{
    int arr[] = { 10, 20, 60, 50, 30, 40 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    cout << partition(arr, n, k) << endl;

    return 0;
}
```

Java

```
 // Java Program for The painter's partition problem
import java.util.*;
import java.io.*;

class GFG
{
```

```
// function to calculate sum between two indices
// in array
static int sum(int arr[], int from, int to)
{
    int total = 0;
    for (int i = from; i <= to; i++)
        total += arr[i];
    return total;
}

// for n boards and k partitions
static int partition(int arr[], int n, int k)
{
    // base cases
    if (k == 1) // one partition
        return sum(arr, 0, n - 1);
    if (n == 1)  // one board
        return arr[0];

    int best = Integer.MAX_VALUE;

    // find minimum of all possible maximum
    // k-1 partitions to the left of arr[i],
    // with i elements, put k-1 th divider
    // between arr[i-1] & arr[i] to get k-th
    // partition
    for (int i = 1; i <= n; i++)
        best = Math.min(best, Math.max(partition(arr, i, k - 1),
                                sum(arr, i, n - 1)));

    return best;
}

// Driver code
public static void main(String args[])
{
 int arr[] = { 10, 20, 60, 50, 30, 40 };

    // Calculate size of array.
    int n = arr.length;
        int k = 3;
 System.out.println(partition(arr, n, k));
}
}

// This code is contributed by Sahil_Bansall
```

C#

```
 // C# Program for The painter's partition problem
using System;

class GFG {

// function to calculate sum
// between two indices in array
static int sum(int []arr, int from, int to)
{
    int total = 0;
    for (int i = from; i <= to; i++)
        total += arr[i];
    return total;
}

// for n boards and k partitions
static int partition(int []arr, int n, int k)
{
    // base cases
    if (k == 1) // one partition
        return sum(arr, 0, n - 1);

    if (n == 1) // one board
        return arr[0];

    int best = int.MaxValue;

    // find minimum of all possible maximum
    // k-1 partitions to the left of arr[i],
    // with i elements, put k-1 th divider
    // between arr[i-1] & arr[i] to get k-th
    // partition
    for (int i = 1; i <= n; i++)
        best = Math.Min(best, Math.Max(partition(arr, i, k - 1),
                                       sum(arr, i, n - 1)));

    return best;
}

// Driver code
public static void Main()
{
    int []arr = {10, 20, 60, 50, 30, 40};

    // Calculate size of array.
    int n = arr.Length;
    int k = 3;
```

```
    // Function calling
    Console.WriteLine(partition(arr, n, k));
}
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// PHP program for The
// painter's partition problem

// function to calculate sum
// between two indices in array
function sum($arr, $from, $to)
{
    $total = 0;
    for ($i = $from; $i <= $to; $i++)
        $total += $arr[$i];
    return $total;
}

// for n boards
// and k partitions
function partition($arr, $n, $k)
{
    // base cases
    if ($k == 1) // one partition
        return sum($arr, 0, $n - 1);
    if ($n == 1) // one board
        return $arr[0];

    $best = PHP_INT_MAX;

    // find minimum of all possible
    // maximum k-1 partitions to the
    // left of arr[i], with i elements,
    // put k-1 th divider between
    // arr[i-1] & arr[i] to get k-th
    // partition
    for ($i = 1; $i <= $n; $i++)
        $best = min($best,
                max(partition($arr, $i, $k - 1),
                          sum($arr, $i, $n - 1)));

    return $best;
}
```

```
// Driver Code
$arr = array(10, 20, 60,
             50, 30, 40);
$n = sizeof($arr);
$k = 3;
echo partition($arr, $n, $k), "\n";

// This code is contributed by ajit
?>
```

Output :

```
90
```

The **time complexity** of the above solution is exponential.

2) Overlapping subproblems:
Following is the partial recursion tree for T(4, 3) in above equation.

```
      T(4, 3)
     /    /    \ ..
T(1, 2)  T(2, 2) T(3, 2)
          /..      /..
      T(1, 1)    T(1, 1)
```

We can observe that many subproblems like T(1, 1) in the above problem are being solved again and again. Because of these two properties of this problem, we can solve it using **dynamic programming**, either by top down memoized method or bottom up tabular method. Following is the bottom up tabular implementation:

C++

```
 // A DP based CPP program for painter's partition problem
#include <climits>
#include <iostream>
using namespace std;

// function to calculate sum between two indices
// in array
int sum(int arr[], int from, int to)
{
    int total = 0;
    for (int i = from; i <= to; i++)
        total += arr[i];
    return total;
```

```
}

// bottom up tabular dp
int findMax(int arr[], int n, int k)
{
    // initialize table
    int dp[k + 1][n + 1] = { 0 };

    // base cases
    // k=1
    for (int i = 1; i <= n; i++)
        dp[1][i] = sum(arr, 0, i - 1);

    // n=1
    for (int i = 1; i <= k; i++)
        dp[i][1] = arr[0];

    // 2 to k partitions
    for (int i = 2; i <= k; i++) { // 2 to n boards
        for (int j = 2; j <= n; j++) {

            // track minimum
            int best = INT_MAX;

            // i-1 th separator before position arr[p=1..j]
            for (int p = 1; p <= j; p++)
                best = min(best, max(dp[i - 1][p],
                                sum(arr, p, j - 1)));

            dp[i][j] = best;
        }
    }

    // required
    return dp[k][n];
}

// driver function
int main()
{
    int arr[] = { 10, 20, 60, 50, 30, 40 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    cout << findMax(arr, n, k) << endl;
    return 0;
}
```

Java

```
 // A DP based Java program for
// painter's partition problem
import java.util.*;
import java.io.*;

class GFG
{
// function to calculate sum between two indices
// in array
static int sum(int arr[], int from, int to)
{
    int total = 0;
    for (int i = from; i <= to; i++)
        total += arr[i];
    return total;
}

// bottom up tabular dp
static int findMax(int arr[], int n, int k)
{
    // initialize table
    int dp[][] = new int[k+1][n+1];

    // base cases
    // k=1
    for (int i = 1; i <= n; i++)
        dp[1][i] = sum(arr, 0, i - 1);

    // n=1
    for (int i = 1; i <= k; i++)
        dp[i][1] = arr[0];

    // 2 to k partitions
    for (int i = 2; i <= k; i++) { // 2 to n boards
        for (int j = 2; j <= n; j++) {

            // track minimum
            int best = Integer.MAX_VALUE;

            // i-1 th separator before position arr[p=1..j]
            for (int p = 1; p <= j; p++)
                best = Math.min(best, Math.max(dp[i - 1][p],
                               sum(arr, p, j - 1)));

            dp[i][j] = best;
        }
    }
```

```
    // required
    return dp[k][n];
}

// Driver code
public static void main(String args[])
{
 int arr[] = { 10, 20, 60, 50, 30, 40 };

    // Calculate size of array.
    int n = arr.length;
        int k = 3;
 System.out.println(findMax(arr, n, k));
}
}

// This code is contributed by Sahil_Bansall
```

C#

```
 // A DP based C# program for
// painter's partition problem
using System;

class GFG {

// function to calculate sum between
// two indices in array
static int sum(int []arr, int from, int to)
{
    int total = 0;
    for (int i = from; i <= to; i++)
        total += arr[i];
    return total;
}

// bottom up tabular dp
static int findMax(int []arr, int n, int k)
{
    // initialize table
    int [,]dp = new int[k+1,n+1];

    // base cases
    // k=1
    for (int i = 1; i <= n; i++)
        dp[1,i] = sum(arr, 0, i - 1);

    // n=1
```

```
    for (int i = 1; i <= k; i++)
        dp[i,1] = arr[0];

    // 2 to k partitions
    for (int i = 2; i <= k; i++) { // 2 to n boards
        for (int j = 2; j <= n; j++) {

            // track minimum
            int best = int.MaxValue;

            // i-1 th separator before position arr[p=1..j]
            for (int p = 1; p <= j; p++)
                best = Math.Min(best, Math.Max(dp[i - 1,p],
                                sum(arr, p, j - 1)));

            dp[i,j] = best;
        }
    }

    // required
    return dp[k,n];
}

// Driver code
public static void Main()
{
    int []arr = {10, 20, 60, 50, 30, 40};

    // Calculate size of array.
    int n = arr.Length;
    int k = 3;
    Console.WriteLine(findMax(arr, n, k));
}
}

// This code is contributed by vt_m
```

PHP

```
 <?php
// A DP based PHP program for
// painter's partition problem

// function to calculate sum
// between two indices in array
function sum($arr, $from, $to)
{
    $total = 0;
```

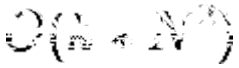
```
    for ($i = $from; $i <= $to; $i++)
        $total += $arr[$i];
    return $total;
}

// bottom up tabular dp
function findMax($arr, $n, $k)
{
    // initialize table
    $dp[$k + 1][$n + 1] = array( 0 );

    // base cases
    // k=1
    for ($i = 1; $i <= $n; $i++)
        $dp[1][$i] = sum($arr, 0,
                        $i - 1);

    // n=1
    for ($i = 1; $i <= $k; $i++)
        $dp[$i][1] = $arr[0];

    // 2 to k partitions
    for ($i = 2; $i <= $k; $i++)
    {
        // 2 to n boards
        for ($j = 2; $j <= $n; $j++)
        {

            // track minimum
            $best = PHP_INT_MAX;

            // i-1 th separator before
            // position arr[p=1..j]
            for ($p = 1; $p <= $j; $p++)
                $best = min($best, max($dp[$i - 1][$p],
                                sum($arr, $p, $j - 1)));

            $dp[$i][$j] = $best;
        }
    }

    // required
    return $dp[$k][$n];
}

// Driver Code
$arr = array (10, 20, 60,
              50, 30, 40 );
```

```
$n = sizeof($arr);
$k = 3;
echo findMax($arr, $n, $k) ,"\n";

// This code is contribted by m_kit
?>
```

Output:

```
90
```

Optimizations:

1) The **time complexity** of the above program is  . It can be easily brought down to  by precomputing the cumulative sums in an array thus avoiding repeated calls to the sum function:

```
int sum[n+1] = {0};

// sum from 1 to i elements of arr
for (int i = 1; i <= n; i++)
  sum[i] = sum[i-1] + arr[i-1];

for (int i = 1; i <= n; i++)
  dp[1][i] = sum[i];

and using it to calculate the result as:
best = min(best, max(dp[i-1][p], sum[j] - sum[p]));
```

2) Though here we consider to divide A into k or fewer partitions, we can observe that the **optimal case** always occurs when we divide A into exactly k partitions. So we can use:

```
for (int i = k-1; i <= n; i++)
   best = min(best, max( partition(arr, i, k-1),
                         sum(arr, i, n-1)));
```

and modify the other implementations accordingly.

Exercise:
1) Can you come up with a solution using binary search which runs in O(N lg N) time? Prerequisite for this: https://www.topcoder.com/community/data-science/data-science-tutorials/binary-search/

References:
https://articles.leetcode.com/the-painters-partition-problem/

Asked in: Google, CodeNation

Improved By : vt_m, jit_t

Source

https://www.geeksforgeeks.org/painters-partition-problem/

Chapter 254

The painter's partition problem | Set 2

The painter's partition problem | Set 2 - GeeksforGeeks

We have to paint n boards of length {A1, A2, .. An}. There are k painters available and each takes 1 unit time to paint 1 unit of board. The problem is to find the minimum time to get this job done under the constraints that any painter will only paint continuous sections of boards, say board {2, 3, 4} or only board {1} or nothing but not board {2, 4, 5}.

Examples :

```
Input : k = 2, A = {10, 10, 10, 10}
Output : 20.
Here we can divide the boards into 2
equal sized partitions, so each painter
gets 20 units of board and the total
time taken is 20.

Input : k = 2, A = {10, 20, 30, 40}
Output : 60.
Here we can divide first 3 boards for
one painter and the last board for
second painter.
```

In the previous post we discussed a dynamic programming based approach having time complexity of $O(k * n^2)$ and $O(k * n)$ extra space.
In this post we will look into a more efficient approach using binary search. We know that the invariant of binary search has two main parts:
* the target value would always be in the searching range.
* the searching range will decrease in each loop so that the termination can be reached.

We also know that the values in this range must be in sorted order. Here our target value is the maximum sum of a contiguous section in the optimal allocation of boards. Now how can we apply binary search for this? We can fix the possible low to high range for the target value and narrow down our search to get the optimal allocation.

We can see that the highest possible value in this range is the sum of all the elementsin the array and this happens when we allot 1 painter all the sections of the board. The lowest possible value of this range is the maximum value of the array max, as in this allocation we can allot max to one painter and divide the other sections such
that the cost of them is less than or equal to max and as close as possible to max. Now if we consider we use x painters in the above scenarios, it is obvious that as the value in the range increases, the value of x decreases and vice-versa. From this we can find the target value when x=k and use a helper function to find x, the minimum number of painters required when the maximum length of section a painter can paint is given.

C++

```
 // CPP program for painter's partition problem
#include <iostream>
using namespace std;

// return the maximum element from the array
int getMax(int arr[], int n)
{
    int max = INT_MIN;
    for (int i = 0; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

// return the sum of the elements in the array
int getSum(int arr[], int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
        total += arr[i];
    return total;
}

// find minimum required painters for given maxlen
// which is the maximum length a painter can paint
int numberOfPainters(int arr[], int n, int maxLen)
{
    int total = 0, numPainters = 1;

    for (int i = 0; i < n; i++) {
        total += arr[i];
```

```
        if (total > maxLen) {

            // for next count
            total = arr[i];
            numPainters++;
        }
    }

    return numPainters;
}

int partition(int arr[], int n, int k)
{
    int lo = getMax(arr, n);
    int hi = getSum(arr, n);

    while (lo < hi) {
        int mid = lo + (hi - lo) / 2;
        int requiredPainters = numberOfPainters(arr, n, mid);

        // find better optimum in lower half
        // here mid is included because we
        // may not get anything better
        if (requiredPainters <= k)
            hi = mid;

        // find better optimum in upper half
        // here mid is excluded because it gives
        // required Painters > k, which is invalid
        else
            lo = mid + 1;
    }

    // required
    return lo;
}

// driver function
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    cout << partition(arr, n, k) << endl;
    return 0;
}
```

Java

```
 // Java Program for painter's partition problem
import java.util.*;
import java.io.*;

class GFG
{
// return the maximum element from the array
static int getMax(int arr[], int n)
{
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

// return the sum of the elements in the array
static int getSum(int arr[], int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
        total += arr[i];
    return total;
}

// find minimum required painters for given maxlen
// which is the maximum length a painter can paint
static int numberOfPainters(int arr[], int n, int maxLen)
{
    int total = 0, numPainters = 1;

    for (int i = 0; i < n; i++) {
        total += arr[i];

        if (total > maxLen) {

            // for next count
            total = arr[i];
            numPainters++;
        }
    }

    return numPainters;
}

static int partition(int arr[], int n, int k)
{
    int lo = getMax(arr, n);
```

```
    int hi = getSum(arr, n);

    while (lo < hi) {
        int mid = lo + (hi - lo) / 2;
        int requiredPainters = numberOfPainters(arr, n, mid);

        // find better optimum in lower half
        // here mid is included because we
        // may not get anything better
        if (requiredPainters <= k)
            hi = mid;

        // find better optimum in upper half
        // here mid is excluded because it gives
        // required Painters > k, which is invalid
        else
            lo = mid + 1;
    }

    // required
    return lo;
}

// Driver code
public static void main(String args[])
{
 int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

    // Calculate size of array.
    int n = arr.length;
        int k = 3;
 System.out.println(partition(arr, n, k));
}
}

// This code is contributed by Sahil_Bansall
```

C#

```
 // C# Program for painter's
// partition problem
using System;

class GFG
{

// return the maximum
// element from the array
```

```
static int getMax(int []arr, int n)
{
    int max = int.MinValue;
    for (int i = 0; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

// return the sum of the
// elements in the array
static int getSum(int []arr, int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
        total += arr[i];
    return total;
}

// find minimum required
// painters for given
// maxlen which is the
// maximum length a painter
// can paint
static int numberOfPainters(int []arr,
                            int n, int maxLen)
{
    int total = 0, numPainters = 1;

    for (int i = 0; i < n; i++)
    {
        total += arr[i];

        if (total > maxLen)
        {

            // for next count
            total = arr[i];
            numPainters++;
        }
    }

    return numPainters;
}

static int partition(int []arr,
                     int n, int k)
{
```

```
    int lo = getMax(arr, n);
    int hi = getSum(arr, n);

    while (lo < hi)
    {
        int mid = lo + (hi - lo) / 2;
        int requiredPainters =
                       numberOfPainters(arr, n, mid);

        // find better optimum in lower
        // half here mid is included
        // because we may not get
        // anything better
        if (requiredPainters <= k)
            hi = mid;

        // find better optimum in upper
        // half here mid is excluded
        // because it gives required
        // Painters > k, which is invalid
        else
            lo = mid + 1;
    }

    // required
    return lo;
}

// Driver code
static public void Main ()
{
    int []arr = {1, 2, 3, 4, 5,
                 6, 7, 8, 9};

    // Calculate size of array.
    int n = arr.Length;
    int k = 3;
    Console.WriteLine(partition(arr, n, k));
}
}

// This code is contributed by ajit
```

PHP

```
 <?php
// PHP program for painter's
// partition problem
```

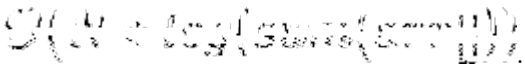
```
// return the maximum
// element from the array
function getMax($arr, $n)
{
    $max = PHP_INT_MIN;
    for ($i = 0; $i < $n; $i++)
        if ($arr[$i] > $max)
            $max = $arr[$i];
    return $max;
}

// return the sum of the
// elements in the array
function getSum($arr, $n)
{
    $total = 0;
    for ($i = 0; $i < $n; $i++)
        $total += $arr[$i];
    return $total;
}

// find minimum required painters
// for given maxlen which is the
// maximum length a painter can paint
function numberOfPainters($arr, $n,
                          $maxLen)
{
    $total = 0; $numPainters = 1;

    for ($i = 0; $i < $n; $i++)
    {
        $total += $arr[$i];

        if ($total > $maxLen)
        {

            // for next count
            $total = $arr[$i];
            $numPainters++;
        }
    }

    return $numPainters;
}

function partition($arr, $n, $k)
{
```

```
    $lo = getMax($arr, $n);
    $hi = getSum($arr, $n);

    while ($lo < $hi)
    {
        $mid = $lo + ($hi - $lo) / 2;
        $requiredPainters =
                    numberOfPainters($arr,
                                     $n, $mid);

        // find better optimum in
        // lower half here mid is
        // included because we may
        // not get anything better
        if ($requiredPainters <= $k)
            $hi = $mid;

        // find better optimum in
        // upper half here mid is
        // excluded because it
        // gives required Painters > k,
        // which is invalid
        else
            $lo = $mid + 1;
    }

    // required
    return floor($lo);
}

// Driver Code
$arr = array(1, 2, 3,
             4, 5, 6,
             7, 8, 9);
$n = sizeof($arr);
$k = 3;

echo partition($arr, $n, $k) , "\n";

// This code is contributed by ajit
?>
```

Output :

```
17
```

For better understanding, please trace the example given in the program in pen and paper.

The time complexity of the above approach is  .

References:
https://articles.leetcode.com/the-painters-partition-problem-part-ii/
https://www.topcoder.com/community/data-science/data-science-tutorials/binary-search/

Asked in: Google, Codenation.

Improved By : jit_t

Source

https://www.geeksforgeeks.org/painters-partition-problem-set-2/

Chapter 255

Third largest element in an array of distinct elements

Third largest element in an array of distinct elements - GeeksforGeeks

Given an array of distinct elements, find third largest element in it.

Example :

```
Input  : arr[] = {1, 14, 2, 16, 10, 20}
Output : The third Largest element is 14

Input  : arr[] = {19, -10, 20, 14, 2, 16, 10}
Output : The third Largest element is 16
```

Method 1 (Simple) Simplest way to solve this question is to first iterate through the array and find first maximum. Store this first maximum as well as its index. Now traverse the whole array finding the second max with the changed condition. Finally traverse the array third time and find the third largest element.

C

```
 // C program to find third Largest element in an array
// of distinct elements
#include <bits/stdc++.h>

void thirdLargest(int arr[], int arr_size)
{
    /* There should be atleast three elements */
    if (arr_size < 3)
    {
        printf(" Invalid Input ");
```

```
        return;
    }

    // Find first largest element
    int first = arr[0];
    for (int i = 1; i < arr_size ; i++)
        if (arr[i] > first)
            first = arr[i];

    // Find second largest element
    int second = INT_MIN;
    for (int i = 0; i < arr_size ; i++)
        if (arr[i] > second && arr[i] < first)
            second = arr[i];

    // Find third largest element
    int third = INT_MIN;
    for (int i = 0; i < arr_size ; i++)
        if (arr[i] > third && arr[i] < second)
            third = arr[i];

    printf("The third Largest element is %d\n", third);
}

/* Driver program to test above function */
int main()
{
    int arr[] = {12, 13, 1, 10, 34, 16};
    int n = sizeof(arr)/sizeof(arr[0]);
    thirdLargest(arr, n);
    return 0;
}
```

Java

```
 // Java program to find third
// Largest element in an array
// of distinct elements

class GFG
{
static void thirdLargest(int arr[],
                         int arr_size)
{
    /* There should be
    atleast three elements */
    if (arr_size < 3)
    {
```

```
        System.out.printf(" Invalid Input ");
        return;
    }

    // Find first
    // largest element
    int first = arr[0];
    for (int i = 1;
             i < arr_size ; i++)
        if (arr[i] > first)
            first = arr[i];

    // Find second
    // largest element
    int second = Integer.MIN_VALUE;
    for (int i = 0;
             i < arr_size ; i++)
        if (arr[i] > second &&
            arr[i] < first)
            second = arr[i];

    // Find third
    // largest element
    int third = Integer.MIN_VALUE;
    for (int i = 0;
             i < arr_size ; i++)
        if (arr[i] > third &&
            arr[i] < second)
            third = arr[i];

    System.out.printf("The third Largest " +
                  "element is %d\n", third);
}

// Driver code
public static void main(String []args)
{
    int arr[] = {12, 13, 1,
                 10, 34, 16};
    int n = arr.length;
    thirdLargest(arr, n);
}
}

// This code is contributed
// by Smitha
```

Python3

```
 # Python 3 program to find
# third Largest element in
# an array of distinct elements
import sys
def thirdLargest(arr, arr_size):

    # There should be
    # atleast three elements
    if (arr_size < 3):

        print(" Invalid Input ")
        return


    # Find first
    # largest element
    first = arr[0]
    for i in range(1, arr_size):
        if (arr[i] > first):
            first = arr[i]

    # Find second
    # largest element
    second = -sys.maxsize
    for i in range(0, arr_size):
        if (arr[i] > second and
            arr[i] < first):
            second = arr[i]

    # Find third
    # largest element
    third = -sys.maxsize
    for i in range(0, arr_size):
        if (arr[i] > third and
            arr[i] < second):
            third = arr[i]

    print("The Third Largest",
          "element is", third)

# Driver Code
arr = [12, 13, 1,
       10, 34, 16]
n = len(arr)
thirdLargest(arr, n)

# This code is contributed
# by Smitha
```

C#

```
 // C# program to find third
// Largest element in an array
// of distinct elements
using System;

class GFG
{
static void thirdLargest(int []arr,
                         int arr_size)
{
    /* There should be
    atleast three elements */
    if (arr_size < 3)
    {
        Console.Write(" Invalid Input ");
        return;
    }

    // Find first
    // largest element
    int first = arr[0];
    for (int i = 1;
             i < arr_size ; i++)
        if (arr[i] > first)
            first = arr[i];

    // Find second
    // largest element
    int second = -int.MaxValue;
    for (int i = 0;
             i < arr_size ; i++)
        if (arr[i] > second &&
            arr[i] < first)
            second = arr[i];

    // Find third
    // largest element
    int third = -int.MaxValue;
    for (int i = 0;
             i < arr_size ; i++)
        if (arr[i] > third &&
            arr[i] < second)
            third = arr[i];

    Console.Write("The third Largest " +
                  "element is "+ third);
```

```
}

// Driver code
public static void Main()
{
    int []arr = {12, 13, 1,
                 10, 34, 16};
    int n = arr.Length;
    thirdLargest(arr, n);
}
}

// This code is contributed by Smitha
```

[/sourcecode]

PHP

```
 <?php
// PHP program to find third
// Largest element in an array
// of distinct elements

function thirdLargest($arr, $arr_size)
{
    /* There should be atleast
    three elements */
    if ($arr_size < 3)
    {
        echo " Invalid Input ";
        return;
    }

    // Find first largest element
    $first = $arr[0];
    for ($i = 1; $i < $arr_size ; $i++)
        if ($arr[$i] > $first)
            $first = $arr[$i];

    // Find second largest element
    $second = PHP_INT_MIN;
    for ($i = 0; $i < $arr_size ; $i++)
        if ($arr[$i] > $second &&
               $arr[$i] < $first)
            $second = $arr[$i];

    // Find third largest element
    $third = PHP_INT_MIN;
```

```
    for ($i = 0; $i < $arr_size ; $i++)
        if ($arr[$i] > $third &&
              $arr[$i] < $second)
            $third = $arr[$i];

    echo "The third Largest element is ",
                              $third,"\n";
}

// Driver Code
$arr = array(12, 13, 1,
             10, 34, 16);
$n = sizeof($arr);
thirdLargest($arr, $n);

// This code is contributed by m_kit
?>
```

Output :

```
The third Largest element is 13
```

Method 2 In this method, we need not to iterate array three times. We can find third largest in one traversal only.

1. Initialize first = a[0] and second = -INF, third = -INF
2. Iterate the array and compare each element with first.
   - If a[i] is greater than first then update all first, second and third:
     third = second
     second = first
     first = arr[i]
   - Else compare arr[i] with second, if its greater than second, then update:
     third = second
     second = arr[i]
   - Else compare arr[i] with third, if its greater than third, then update:
     third = arr[i]
3. Return third

C

```
 // C program to find third Largest element in an array
#include <bits/stdc++.h>

void thirdLargest(int arr[], int arr_size)
{
```

```
    /* There should be atleast three elements */
    if (arr_size < 3)
    {
        printf(" Invalid Input ");
        return;
    }

    // Initialize first, second and third Largest element
    int first = arr[0], second = INT_MIN, third = INT_MIN;

    // Traverse array elements to find the third Largest
    for (int i = 1; i < arr_size ; i ++)
    {
        /* If current element is greater than first,
           then update first, second and third */
        if (arr[i] > first)
        {
            third  = second;
            second = first;
            first  = arr[i];
        }

        /* If arr[i] is in between first and second */
        else if (arr[i] > second)
        {
            third = second;
            second = arr[i];
        }

        /* If arr[i] is in between second and third */
        else if (arr[i] > third)
            third = arr[i];
    }

    printf("The third Largest element is %d\n", third);
}

/* Driver program to test above function */
int main()
{
    int arr[] = {12, 13, 1, 10, 34, 16};
    int n = sizeof(arr)/sizeof(arr[0]);
    thirdLargest(arr, n);
    return 0;
}
```

Python 3

```
 # Python 3 program to find
# third Largest element in
# an array
import sys
def thirdLargest(arr, arr_size):

    # There should be
    # atleast three elements
    if (arr_size < 3):

        print(" Invalid Input ")
        return

    # Initialize first, second
    # and third Largest element
    first = arr[0]
    second = -sys.maxsize
    third = -sys.maxsize

    # Traverse array elements
    # to find the third Largest
    for i in range(1, arr_size):

        # If current element is
        # greater than first,
        # then update first,
        # second and third
        if (arr[i] > first):

            third = second
            second = first
            first = arr[i]


        # If arr[i] is in between
        # first and second
        elif (arr[i] > second):

            third = second
            second = arr[i]

        # If arr[i] is in between
        # second and third
        elif (arr[i] > third):
            third = arr[i]

    print("The third Largest" ,
                  "element is", third)
```

```
# Driver Code
arr = [12, 13, 1,
       10, 34, 16]
n = len(arr)
thirdLargest(arr, n)

# This code is contributed
# by Smitha
```

PHP

```
 <?php
// PHP program to find third
// Largest element in an array

function thirdLargest($arr, $arr_size)
{
    /* There should be atleast
       three elements */
    if ($arr_size < 3)
    {
        echo " Invalid Input ";
        return;
    }

    // Initialize first, second and
    // third Largest element
    $first = $arr[0];
    $second = PHP_INT_MIN;
    $third = PHP_INT_MIN;

    // Traverse array elements to
    // find the third Largest
    for ($i = 1; $i < $arr_size ; $i ++)
    {
        /* If current element is greater
        than first, then update first,
        second and third */
        if ($arr[$i] > $first)
        {
            $third = $second;
            $second = $first;
            $first = $arr[$i];
        }

        /* If arr[i] is in between
        first and second */
```

```
        else if ($arr[$i] > $second)
        {
            $third = $second;
            $second = $arr[$i];
        }

        /* If arr[i] is in between
        second and third */
        else if ($arr[$i] > $third)
            $third = $arr[$i];
    }

    echo "The third Largest element is ",
                                    $third;
}

// Driver Code
$arr = array (12, 13, 1,
              10, 34, 16);
$n = sizeof($arr);
thirdLargest($arr, $n);

// This code is contributed by jit_t
?>
```

Output :

```
The third Largest element is 13
```

Exercise :
Extend the above solution to find third largest when array may have duplicates. For example, if the input array is {10, 5, 15, 5, 15, 10, 1, 1}, then output should be 5. The extended solution should also work in one traversal.

Related Articles:

1. Find the smallest and second smallest element in an array
2. k largest(or smallest) elements in an array
3. K'th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time)

Improved By : jit_t, Smitha Dinesh Semwal

Source

https://www.geeksforgeeks.org/third-largest-element-array-distinct-elements/

Chapter 256

Triplets in array with absolute difference less than k

Triplets in array with absolute difference less than k - GeeksforGeeks

Given an array **A[]** of **n** elements and an integer **k**. The task is to find the number of triplet (x, y, k), where 0 <= x, y, k < n and x, y, k are the index in the array A[] such that:
|A[x] – A[y]| <= k
|A[y] – A[z]| <= k
|A[z] – A[x]| <= k

Examples:

```
Input : A[] = { 1, 1, 2, 2, 3 }, k = 1
Output : 5
(0, 1, 2), (0, 1, 3), (0, 2, 3), (1, 2, 3),
(2, 3, 4) are the triplet whose element will
satisfy the above three condition.

Input : A[] = { 1, 2, 3 }, k = 1
Output : 1
```

A **simple solution** is to run three nested loops and count triplets with given constraints.

An **efficient solution** is based on the fact rearranging the elements in the array does not affect the answer because basically, we want index x, y, z to be in increasing order, not A[x], A[y], A[z] to be sorted. Suppose, A[x] is at index y, A[y] at z, A[z] at x, still we can pick the triplet (x, y, z) because the condition of difference between any two elements to be less k still stands.
Now, to calculate the number of triplet indices (x, y, z) which satisfies the above condition, we will sort the given array. Now, for each element A[i], i >= 2, we will find the lower bound index of A[i] – k, say lb. Now, observe all the element between index lb and i are less than

A[i] and the difference between any teo elements will be less than or equal to k. So, element at index i can be treated as index z and we can choose any two element from lb to i – 1. So, this will increase the count of the triplet by $^{i - lb}C^2$.

Below is the implementation of this approach:

C++

```
 // CPP program to count triplets with difference less
// than k.
#include <bits/stdc++.h>
using namespace std;

// Return the lower bound i.e smallest index of
// element having value greater or equal to value
int binary_lower(int value, int arr[], int n)
{
    int start = 0;
    int end = n - 1;
    int ans = -1;
    int mid;

    while (start <= end) {
        mid = (start + end) / 2;
        if (arr[mid] >= value) {
            end = mid - 1;
            ans = mid;
        }
        else {
            start = mid + 1;
        }
    }
    return ans;
}

// Return the number of triplet indices satisfies
// the three constraints
int countTriplet(int arr[], int n, int k)
{
    int count = 0;

    // sort the array
    sort(arr, arr + n);

    // for each element from index 2 to n - 1.
    for (int i = 2; i < n; i++) {

        // finding the lower bound of arr[i] - k.
```

```
        int cur = binary_lower(arr[i] - k, arr, n);

        // If there are at least two elements between
        // lower bound and current element.
        if (cur <= i - 2) {

            // increment the count by lb - i C 2.
            count += ((i - cur) * (i - cur - 1)) / 2;
        }
    }

    return count;
}
int main()
{
    int arr[] = { 1, 1, 2, 2, 3 };
    int k = 1;
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << countTriplet(arr, n, k) << endl;
    return 0;
}
```

Java

```
 // Java program to count triplets
// with difference less than k.
import java.io.*;
import java .util.*;

class GFG
{

// Return the lower bound i.e
// smallest index of element
// having value greater or
// equal to value
static int binary_lower(int value,
                        int arr[],
                        int n)
{
    int start = 0;
    int end = n - 1;
    int ans = -1;
    int mid;

    while (start <= end)
    {
```

```
        mid = (start + end) / 2;
        if (arr[mid] >= value)
        {
            end = mid - 1;
            ans = mid;
        }
        else
        {
            start = mid + 1;
        }
    }
    return ans;
}

// Return the number of
// triplet indices satisfies
// the three constraints
static int countTriplet(int arr[],
                        int n, int k)
{
    int count = 0;

    // sort the array
    Arrays.sort(arr);

    // for each element from
    // index 2 to n - 1.
    for (int i = 2; i < n; i++)
    {

        // finding the lower
        // bound of arr[i] - k.
        int cur = binary_lower(arr[i] - k,
                               arr, n);

        // If there are at least two
        // elements between lower
        // bound and current element.
        if (cur <= i - 2)
        {

            // increment the count
            // by lb - i C 2.
            count += ((i - cur) *
                      (i - cur - 1)) / 2;
        }
    }
    return count;
```

```
}

// Driver Code
public static void main (String[] args)
{
    int arr[] = {1, 1, 2, 2, 3};
    int k = 1;
    int n = arr.length;

    System.out.println(countTriplet(arr, n, k));
}
}

// This code is contributed by anuj_67.
```

C#

```
 // C# program to count triplets
// with difference less than k.
using System;

class GFG
{

// Return the lower bound i.e
// smallest index of element
// having value greater or
// equal to value
static int binary_lower(int value,
                        int []arr,
                        int n)
{
    int start = 0;
    int end = n - 1;
    int ans = -1;
    int mid;

    while (start <= end)
    {
        mid = (start + end) / 2;
        if (arr[mid] >= value)
        {
            end = mid - 1;
            ans = mid;
        }
        else
        {
            start = mid + 1;
```

```
        }
    }
    return ans;
}

// Return the number of
// triplet indices satisfies
// the three constraints
static int countTriplet(int []arr,
                        int n, int k)
{
    int count = 0;

    // sort the array
    Array.Sort(arr);

    // for each element from
    // index 2 to n - 1.
    for (int i = 2; i < n; i++)
    {

        // finding the lower
        // bound of arr[i] - k.
        int cur = binary_lower(arr[i] - k,
                               arr, n);

        // If there are at least two
        // elements between lower
        // bound and current element.
        if (cur <= i - 2)
        {

            // increment the count
            // by lb - i C 2.
            count += ((i - cur) *
                      (i - cur - 1)) / 2;
        }
    }
    return count;
}

// Driver Code
public static void Main ()
{
    int []arr = {1, 1, 2, 2, 3};
    int k = 1;
    int n = arr.Length;
```

```
    Console.WriteLine(countTriplet(arr, n, k));
}
}

// This code is contributed by anuj_67.
```

Output:

```
5
```

Complexity: O(nlogn)

Improved By : vt_m

Source

https://www.geeksforgeeks.org/triplets-array-absolute-difference-less-k/

Chapter 257

Two Pointers Technique

Two Pointers Technique - GeeksforGeeks

Two pointer is really an easy and effective technique which is typically used for searching pairs in a sorted arrays.

Given a sorted array A (sorted in ascending order), having N integers, find if there exists any pair of elements (A[i], A[j]) such that their sum is equal to X.

Let's see the **naive solution**.

```
 // Naive solution to find if there is a
// pair in A[0..N-1] with given sum.

bool isPairSum(A[], N, X)
{
   for (i = 0; i < N; i++) {
       for (j = 0; j < N; j++) {
           if (A[i] + A[j] == X)
               return true; // pair exists

           if (A[i] + A[j] > X)
               break; // as the array is sorted
       }
   }

   // No pair found with given sum.
   return false;
}
```

The time complexity of this solution is $\mathbf{O(n^2)}$.

Now let's see how the two pointer technique works. We take two pointers, one representing the first element and other representing the last element of the array, and then we add the

values kept at both the pointers. If their sum is smaller than X then we shift the left pointer to right or if their sum is greater than X then we shift the right pointer to left, in order to get closer to the sum. We keep moving the pointers until we get the sum as X.

```
 // Two pointer technique based solution to find
// if there is a pair in A[0..N-1] with given sum.
bool isPairSum(A[], N, X)
{
    // represents first pointer
    int i = 0;

    // represents second pointer
    int j = N - 1;

    while (i < j) {

        // If we find a pair
        if (A[i] + A[j] == X)
            return true;

        // If sum of elements at current
        // pointers is less, we move towards
        // higher values by doing i++
        else if (A[i] + A[j] < X)
            i++;

        // If sum of elements at current
        // pointers is more, we move towards
        // lower values by doing i++
        else
            j--;
    }
    return false;
}
```

Illustration :

```
A[] = {10, 20, 35, 50, 75, 80}
X = 70
i = 0
j - 5

A[i] + A[j] = 10 + 80 = 90.
Since A[i] + A[j] > X, j--
i = 0, j = 4

A[i] + A[j] = 10 + 75 = 85
Since A[i] + A[j] > X, j--
i = 0
j = 3

A[i] + A[j] = 10 + 50 = 60
Since A[i] + A[j] < X, i++
i = 1
j = 3

A[i] + A[j] = 20 + 50 = 70
We found the pair!
```

The above solution works in **O(n)**

How does this work?
The algorithm basically uses the fact that input array is sorted. We start sum of extreme values (smallest and largest) and conditionally move both pointers. We move left pointer i when sum of A[i] and A[j[is less than X. We do not miss any pair because sum is already smaller than X. Same logic applies for right pointer j.

More problems based on two pointer technique.

- Find the closest pair from two sorted arrays
- Find the pair in array whose sum is closest to x
- Find all triplets with zero sum
- Find a triplet that sum to a given value
- Find a triplet such that sum of two equals to third element
- Find four elements that sum to a given value

Source

https://www.geeksforgeeks.org/two-pointers-technique/

Chapter 258

Two elements whose sum is closest to zero

Two elements whose sum is closest to zero - GeeksforGeeks

Question: An Array of integers is given, both +ve and -ve. You need to find the two elements such that their sum is closest to zero.

For the below array, program should print -80 and 85.

METHOD 1 (Simple)
For each element, find the sum of it with every other element in the array and compare sums. Finally, return the minimum sum.

Implementation:

C

```
 # include <stdio.h>
# include <stdlib.h> /* for abs() */
# include <math.h>
void minAbsSumPair(int arr[], int arr_size)
{
  int inv_count = 0;
  int l, r, min_sum, sum, min_l, min_r;

  /* Array should have at least two elements*/
  if(arr_size < 2)
  {
    printf("Invalid Input");
    return;
  }

  /* Initialization of values */
```

```
  min_l = 0;
  min_r = 1;
  min_sum = arr[0] + arr[1];

  for(l = 0; l < arr_size - 1; l++)
  {
    for(r = l+1; r < arr_size; r++)
    {
      sum = arr[l] + arr[r];
      if(abs(min_sum) > abs(sum))
      {
        min_sum = sum;
        min_l = l;
        min_r = r;
      }
    }
  }

  printf(" The two elements whose sum is minimum are %d and %d",
         arr[min_l], arr[min_r]);
}

/* Driver program to test above function */
int main()
{
  int arr[] = {1, 60, -10, 70, -80, 85};
  minAbsSumPair(arr, 6);
  getchar();
  return 0;
}
```

Java

```
 import java.util.*;
import java.lang.*;
class Main
{
    static void minAbsSumPair(int arr[], int arr_size)
    {
      int inv_count = 0;
      int l, r, min_sum, sum, min_l, min_r;

      /* Array should have at least two elements*/
      if(arr_size < 2)
      {
        System.out.println("Invalid Input");
        return;
      }
```

```
      /* Initialization of values */
      min_l = 0;
      min_r = 1;
      min_sum = arr[0] + arr[1];

      for(l = 0; l < arr_size - 1; l++)
      {
        for(r = l+1; r < arr_size; r++)
        {
          sum = arr[l] + arr[r];
          if(Math.abs(min_sum) > Math.abs(sum))
          {
            min_sum = sum;
            min_l = l;
            min_r = r;
          }
        }
      }

      System.out.println(" The two elements whose "+
                              "sum is minimum are "+
                         arr[min_l]+ " and "+arr[min_r]);
    }

    // main function
    public static void main (String[] args)
    {
        int arr[] = {1, 60, -10, 70, -80, 85};
        minAbsSumPair(arr, 6);
    }

}
```

Python3

```
 # Python3 code to find Two elements
# whose sum is closest to zero

def minAbsSumPair(arr,arr_size):
    inv_count = 0

    # Array should have at least
    # two elements
    if arr_size < 2:
        print("Invalid Input")
        return
```

```
    # Initialization of values
    min_l = 0
    min_r = 1
    min_sum = arr[0] + arr[1]
    for l in range (0, arr_size - 1):
        for r in range (l + 1, arr_size):
            sum = arr[l] + arr[r]
            if abs(min_sum) > abs(sum):
                min_sum = sum
                min_l = l
                min_r = r

    print("The two elements whose sum is minimum are",
            arr[min_l], "and ", arr[min_r])

# Driver program to test above function
arr = [1, 60, -10, 70, -80, 85]

minAbsSumPair(arr, 6);

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
 // C# code to find Two elements
// whose sum is closest to zero
using System;

class GFG
{
static void minAbsSumPair(int []arr,
                          int arr_size)
    {

    int l, r, min_sum, sum, min_l, min_r;

    /* Array should have at least two elements*/
    if (arr_size < 2)
    {
        Console.Write("Invalid Input");
        return;
    }

    /* Initialization of values */
    min_l = 0;
    min_r = 1;
    min_sum = arr[0] + arr[1];
```

```
    for (l = 0; l < arr_size - 1; l++)
    {
        for (r = l+1; r < arr_size; r++)
        {
            sum = arr[l] + arr[r];
            if (Math.Abs(min_sum) > Math.Abs(sum))
            {
                min_sum = sum;
                min_l = l;
                min_r = r;
            }
        }
    }

    Console.Write(" The two elements whose "+
                        "sum is minimum are "+
                    arr[min_l]+ " and "+arr[min_r]);
    }

    // main function
    public static void Main ()
    {
        int []arr = {1, 60, -10, 70, -80, 85};

        minAbsSumPair(arr, 6);
    }

}

// This code is contributed by Sam007
```

PHP

```
 <?php
// PHP program to find the Two elements
// whose sum is closest to zero

function minAbsSumPair($arr, $arr_size)
{
    $inv_count = 0;

    /* Array should have at
    least two elements*/
    if($arr_size < 2)
    {
        echo "Invalid Input";
        return;
    }
```

```
    /* Initialization of values */
    $min_l = 0;
    $min_r = 1;
    $min_sum = $arr[0] + $arr[1];

    for($l = 0; $l < $arr_size - 1; $l++)
    {
        for($r = $l+1; $r < $arr_size; $r++)
        {
        $sum = $arr[$l] + $arr[$r];
        if(abs($min_sum) > abs($sum))
        {
            $min_sum = $sum;
            $min_l = $l;
            $min_r = $r;
        }
        }
    }

    echo "The two elements whose sum is minimum are "
            .$arr[$min_l]." and ". $arr[$min_r];

}

// Driver Code
$arr = array(1, 60, -10, 70, -80, 85);
minAbsSumPair($arr, 6);

// This code is contributed by Sam007
?>
```

Output:

```
The two elements whose sum is minimum are -80 and 85
```

Time complexity: O(n^2)

METHOD 2 (Use Sorting)
Thanks to baskin for suggesting this approach. We recommend to read this post for background of this approach.

Algorithm
1) Sort all the elements of the input array.
2) Use two index variables l and r to traverse from left and right ends respectively. Initialize l as 0 and r as n-1.
3) sum = a[l] + a[r]

4) If sum is -ve, then l++
5) If sum is +ve, then r–
6) Keep track of abs min sum.
7) Repeat steps 3, 4, 5 and 6 while l < r

Implementation

C

```
 # include <stdio.h>
# include <math.h>
# include <limits.h>

void quickSort(int *, int, int);

/* Function to print pair of elements having minimum sum */
void minAbsSumPair(int arr[], int n)
{
  // Variables to keep track of current sum and minimum sum
  int sum, min_sum = INT_MAX;

  // left and right index variables
  int l = 0, r = n-1;

  // variable to keep track of the left and right pair for min_sum
  int min_l = l, min_r = n-1;

  /* Array should have at least two elements*/
  if(n < 2)
  {
    printf("Invalid Input");
    return;
  }

  /* Sort the elements */
  quickSort(arr, l, r);

  while(l < r)
  {
    sum = arr[l] + arr[r];

    /*If abs(sum) is less then update the result items*/
    if(abs(sum) < abs(min_sum))
    {
      min_sum = sum;
      min_l = l;
      min_r = r;
    }
```

```
    if(sum < 0)
      l++;
    else
      r--;
  }

  printf(" The two elements whose sum is minimum are %d and %d",
         arr[min_l], arr[min_r]);
}

/* Driver program to test above function */
int main()
{
  int arr[] = {1, 60, -10, 70, -80, 85};
  int n = sizeof(arr)/sizeof(arr[0]);
  minAbsSumPair(arr, n);
  getchar();
  return 0;
}

/* FOLLOWING FUNCTIONS ARE ONLY FOR SORTING
    PURPOSE */
void exchange(int *a, int *b)
{
  int temp;
  temp = *a;
  *a   = *b;
  *b   = temp;
}

int partition(int arr[], int si, int ei)
{
  int x = arr[ei];
  int i = (si - 1);
  int j;

  for (j = si; j <= ei - 1; j++)
  {
    if(arr[j] <= x)
    {
      i++;
      exchange(&arr[i], &arr[j]);
    }
  }

  exchange (&arr[i + 1], &arr[ei]);
  return (i + 1);
}
```

```
/* Implementation of Quick Sort
arr[] --> Array to be sorted
si  --> Starting index
ei  --> Ending index
*/
void quickSort(int arr[], int si, int ei)
{
  int pi;    /* Partitioning index */
  if(si < ei)
  {
    pi = partition(arr, si, ei);
    quickSort(arr, si, pi - 1);
    quickSort(arr, pi + 1, ei);
  }
}
```

Java

```
 import java.util.*;
import java.lang.*;
class Main
{
    static void minAbsSumPair(int arr[], int n)
    {
      // Variables to keep track of current sum and minimum sum
      int sum, min_sum = 999999;

      // left and right index variables
      int l = 0, r = n-1;

      // variable to keep track of the left and right pair for min_sum
      int min_l = l, min_r = n-1;

      /* Array should have at least two elements*/
      if(n < 2)
      {
        System.out.println("Invalid Input");
        return;
      }

      /* Sort the elements */
      sort(arr, l, r);

      while(l < r)
      {
        sum = arr[l] + arr[r];
```

```
    /*If abs(sum) is less then update the result items*/
    if(Math.abs(sum) < Math.abs(min_sum))
    {
      min_sum = sum;
      min_l = l;
      min_r = r;
    }
    if(sum < 0)
      l++;
    else
      r--;
  }


  System.out.println(" The two elements whose "+
                          "sum is minimum are "+
                   arr[min_l]+ " and "+arr[min_r]);
}

// main function
public static void main (String[] args)
{
    int arr[] = {1, 60, -10, 70, -80, 85};
    int n = arr.length;
    minAbsSumPair(arr, n);
}

/* Functions for QuickSort */

/* This function takes last element as pivot,
   places the pivot element at its correct
   position in sorted array, and places all
   smaller (smaller than pivot) to left of
   pivot and all greater elements to right
   of pivot */
static int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low-1); // index of smaller element
    for (int j=low; j<high; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;

            // swap arr[i] and arr[j]
```

```
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }


    /* The main function that implements QuickSort()
      arr[] --> Array to be sorted,
      low  --> Starting index,
      high  --> Ending index */
    static void sort(int arr[], int low, int high)
    {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi] is
              now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }
}
```

C#

```
 using System;

class GFG
{
    static void minAbsSumPair(int []arr ,int n)
    {
        // Variables to keep track
        // of current sum and minimum sum
        int sum, min_sum = 999999;

        // left and right index variables
```

```
    int l = 0, r = n-1;

    // variable to keep track of the left
    // and right pair for min_sum
    int min_l = l, min_r = n-1;

    /* Array should have at least two elements*/
    if (n < 2)
    {
        Console.Write("Invalid Input");
        return;
    }

    /* Sort the elements */
    sort(arr, l, r);

    while(l < r)
    {
        sum = arr[l] + arr[r];

        /*If abs(sum) is less then update the result items*/
        if (Math.Abs(sum) < Math.Abs(min_sum))
        {
            min_sum = sum;
            min_l = l;
            min_r = r;
        }
        if (sum < 0)
            l++;
        else
            r--;
    }

    Console.Write(" The two elements whose " +
                            "sum is minimum are " +
                        arr[min_l]+ " and " + arr[min_r]);
}

// driver code
public static void Main ()
{
    int []arr = {1, 60, -10, 70, -80, 85};
    int n = arr.Length;

    minAbsSumPair(arr, n);
}

/* Functions for QuickSort */
```

```
/* This function takes last element as pivot,
places the pivot element at its correct
position in sorted array, and places all
smaller (smaller than pivot) to left of
pivot and all greater elements to right
of pivot */
static int partition(int []arr, int low, int high)
{
    int pivot = arr[high];
    int i = (low-1); // index of smaller element

    for (int j = low; j < high; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;

            // swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    // swap arr[i+1] and arr[high] (or pivot)
    int temp1 = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp1;

    return i+1;
}


/* The main function that implements QuickSort()
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
static void sort(int []arr, int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is
        now at right place */
        int pi = partition(arr, low, high);
```

```
            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }
}

// This code is contributed by Sam007
```

Output:

```
The two elements whose sum is minimum are -80 and 85
```

Time Complexity: complexity to sort + complexity of finding the optimum pair = O(nlogn) + O(n) = O(nlogn)

Asked by Vineet.

Improved By : Sam007

Source

https://www.geeksforgeeks.org/two-elements-whose-sum-is-closest-to-zero/

Chapter 259

Unbounded Binary Search Example (Find the point where a monotonically increasing function becomes positive first time)

Unbounded Binary Search Example (Find the point where a monotonically increasing function becomes positive first time) - GeeksforGeeks

Given a function 'int f(unsigned int x)' which takes a **non-negative integer** 'x' as input and returns an **integer** as output. The function is monotonically increasing with respect to value of x, i.e., the value of f(x+1) is greater than f(x) for every input x. Find the value 'n' where f() becomes positive for the first time. Since f() is monotonically increasing, values of f(n+1), f(n+2),… must be positive and values of f(n-2), f(n-3), .. must be negative.
Find n in O(logn) time, you may assume that f(x) can be evaluated in O(1) time for any input x.

A **simple solution** is to start from i equals to 0 and one by one calculate value of f(i) for 1, 2, 3, 4 .. etc until we find a positive f(i). This works, but takes O(n) time.

Can we apply Binary Search to find n in O(Logn) time? We can't directly apply Binary Search as we don't have an upper limit or high index. The idea is to do repeated doubling until we find a positive value, i.e., check values of f() for following values until f(i) becomes positive.

```
f(0)
f(1)
f(2)
```

```
  f(4)
  f(8)
  f(16)
  f(32)
  ....
  ....
  f(high)
Let 'high' be the value of i when f() becomes positive for first time.
```

Can we apply Binary Search to find n after finding 'high'? We can apply Binary Search now, we can use 'high/2' as low and 'high' as high indexes in binary search. The result n must lie between 'high/2' and 'high'.

Number of steps for finding 'high' is O(Logn). So we can find 'high' in O(Logn) time. What about time taken by Binary Search between high/2 and high? The value of 'high' must be less than 2*n. The number of elements between high/2 and high must be O(n). Therefore, time complexity of Binary Search is O(Logn) and overall time complexity is 2*O(Logn) which is O(Logn).

C

```
 #include <stdio.h>
int binarySearch(int low, int high); // prototype

// Let's take an example function as f(x) = x^2 - 10*x - 20
// Note that f(x) can be any monotonocally increasing function
int f(int x) { return (x*x - 10*x - 20); }

// Returns the value x where above function f() becomes positive
// first time.
int findFirstPositive()
{
    // When first value itself is positive
    if (f(0) > 0)
        return 0;

    // Find 'high' for binary search by repeated doubling
    int i = 1;
    while (f(i) <= 0)
        i = i*2;

    //  Call binary search
    return binarySearch(i/2, i);
}

// Searches first positive value of f(i) where low <= i <= high
int binarySearch(int low, int high)
{
```

```
    if (high >= low)
    {
        int mid = low + (high - low)/2; /* mid = (low + high)/2 */

        // If f(mid) is greater than 0 and one of the following two
        // conditions is true:
        // a) mid is equal to low
        // b) f(mid-1) is negative
        if (f(mid) > 0 && (mid == low || f(mid-1) <= 0))
            return mid;

        // If f(mid) is smaller than or equal to 0
        if (f(mid) <= 0)
            return binarySearch((mid + 1), high);
        else // f(mid) > 0
            return binarySearch(low, (mid -1));
    }

    /* Return -1 if there is no positive value in given range */
    return -1;
}

/* Driver program to check above functions */
int main()
{
    printf("The value n where f() becomes positive first is %d",
           findFirstPositive());
    return 0;
}
```

Java

```
 // Java program for Binary Search
import java.util.*;

class Binary
{
    public static int f(int x)
    { return (x*x - 10*x - 20); }

    // Returns the value x where above
    // function f() becomes positive
    // first time.
    public static int findFirstPositive()
    {
        // When first value itself is positive
        if (f(0) > 0)
            return 0;
```

```
        // Find 'high' for binary search
        // by repeated doubling
        int i = 1;
        while (f(i) <= 0)
            i = i * 2;

        // Call binary search
        return binarySearch(i / 2, i);
    }

    // Searches first positive value of
    // f(i) where low <= i <= high
    public static int binarySearch(int low, int high)
    {
        if (high >= low)
        {
            /* mid = (low + high)/2 */
            int mid = low + (high - low)/2;

            // If f(mid) is greater than 0 and
            // one of the following two
            // conditions is true:
            // a) mid is equal to low
            // b) f(mid-1) is negative
            if (f(mid) > 0 && (mid == low || f(mid-1) <= 0))
                return mid;

            // If f(mid) is smaller than or equal to 0
            if (f(mid) <= 0)
                return binarySearch((mid + 1), high);
            else // f(mid) > 0
                return binarySearch(low, (mid -1));
        }

        /* Return -1 if there is no positive
        value in given range */
        return -1;
    }

    // driver code
    public static void main(String[] args)
    {
        System.out.print ("The value n where f() "+
                          "becomes positive first is "+
                          findFirstPositive());
    }
}
```

```
// This code is contributed by rishabh_jain
```

Python3

```
 # Python3 program for Unbound Binary search.

# Let's take an example function as
# f(x) = x^2 - 10*x - 20
# Note that f(x) can be any monotonocally
# increasing function
def f(x):
    return (x * x - 10 * x - 20)

# Returns the value x where above function
# f() becomes positive first time.
def findFirstPositive() :

    # When first value itself is positive
    if (f(0) > 0):
        return 0

    # Find 'high' for binary search
    # by repeated doubling
    i = 1
    while (f(i) <= 0) :
        i = i * 2

    # Call binary search
    return binarySearch(i/2, i)

# Searches first positive value of
# f(i) where low <= i <= high
def binarySearch(low, high):
    if (high >= low) :

        # mid = (low + high)/2
        mid = low + (high - low)/2;

        # If f(mid) is greater than 0
        # and one of the following two
        # conditions is true:
        # a) mid is equal to low
        # b) f(mid-1) is negative
        if (f(mid) > 0 and (mid == low or f(mid-1) <= 0)) :
            return mid;

        # If f(mid) is smaller than or equal to 0
```

```
        if (f(mid) <= 0) :
            return binarySearch((mid + 1), high)
        else : # f(mid) > 0
            return binarySearch(low, (mid -1))

    # Return -1 if there is no positive
    # value in given range
    return -1;

# Driver Code
print ("The value n where f() becomes "+
      "positive first is ", findFirstPositive());

# This code is contributed by rishabh_jain
```

C#

```
 // C# program for Binary Search
using System;

class Binary
{
    public static int f(int x)
    {
        return (x*x - 10*x - 20);
    }

    // Returns the value x where above
    // function f() becomes positive
    // first time.
    public static int findFirstPositive()
    {
        // When first value itself is positive
        if (f(0) > 0)
            return 0;

        // Find 'high' for binary search
        // by repeated doubling
        int i = 1;
        while (f(i) <= 0)
            i = i * 2;

        // Call binary search
        return binarySearch(i / 2, i);
    }

    // Searches first positive value of
    // f(i) where low <= i <= high
```

```
    public static int binarySearch(int low, int high)
    {
        if (high >= low)
        {
            /* mid = (low + high)/2 */
            int mid = low + (high - low)/2;

            // If f(mid) is greater than 0 and
            // one of the following two
            // conditions is true:
            // a) mid is equal to low
            // b) f(mid-1) is negative
            if (f(mid) > 0 && (mid == low ||
                            f(mid-1) <= 0))
                return mid;

            // If f(mid) is smaller than or equal to 0
            if (f(mid) <= 0)
                return binarySearch((mid + 1), high);
            else

                // f(mid) > 0
                return binarySearch(low, (mid -1));
        }

        /* Return -1 if there is no positive
        value in given range */
        return -1;
    }

    // Driver code
    public static void Main()
    {
       Console.Write ("The value n where f() " +
                      "becomes positive first is " +
                       findFirstPositive());
    }
}

// This code is contributed by nitin mittal
```

PHP

```
 <?php
// PHP program for Binary Search

// Let's take an example function
// as f(x) = x^2 - 10*x - 20
```

```
// Note that f(x) can be any
// monotonocally increasing function
function f($x)
{
    return ($x * $x - 10 * $x - 20);
}

// Returns the value x where above
// function f() becomes positive
// first time.
function findFirstPositive()
{
    // When first value
    // itself is positive
    if (f(0) > 0)
        return 0;

    // Find 'high' for binary
    // search by repeated doubling
    $i = 1;
    while (f($i) <= 0)
        $i = $i * 2;

    // Call binary search
    return binarySearch(intval($i / 2), $i);
}

// Searches first positive value
// of f(i) where low <= i <= high
function binarySearch($low, $high)
{
    if ($high >= $low)
    {
        /* mid = (low + high)/2 */
        $mid = $low + intval(($high -
                              $low) / 2);

        // If f(mid) is greater than 0
        // and one of the following two
        // conditions is true:
        // a) mid is equal to low
        // b) f(mid-1) is negative
        if (f($mid) > 0 && ($mid == $low ||
                            f($mid - 1) <= 0))
            return $mid;

        // If f(mid) is smaller
        // than or equal to 0
```

```
        if (f($mid) <= 0)
            return binarySearch(($mid + 1), $high);
        else // f(mid) > 0
            return binarySearch($low, ($mid - 1));
    }

    /* Return -1 if there is no
    positive value in given range */
    return -1;
}

// Driver Code
echo "The value n where f() becomes ".
                 "positive first is ".
                 findFirstPositive() ;

// This code is contributed by Sam007
?>
```

Output :

```
The value n where f() becomes positive first is 12
```

Related Article:
Exponential Search

Improved By : nitin mittal, Sam007

Source

https://www.geeksforgeeks.org/find-the-point-where-a-function-becomes-negative/

Chapter 260

Value of k-th index of a series formed by append and insert MEX in middle

Value of k-th index of a series formed by append and insert MEX in middle - GeeksforGeeks

Given two integers, n and k. Initially we have a sequence consisting of a single number 1. We need to consider series formed after n steps. In each step, we append the sequence to itself and insert the **MEX**(minimum excluded)(>0) value of the sequence in the middle. Perform (n-1) steps. Finally, find the value of the k-th index of the resulting sequence.

Example:

```
Input : n = 3, k = 2.
Output: Initially, we have {1}, we have to perform 2 steps.
        1st step :  {1, 2, 1}, since MEX of {1} is 2.
        2nd step:   {1, 2, 1, 3, 1, 2, 1},
                     since MEX of {1, 2, 1} is 3.
        Value of 2nd Index = 2.

Input : n = 4, k = 8.
Output: Perform 3 steps.
        After second step, we have  {1, 2, 1, 3, 1, 2, 1}
        3rd step: {1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3,
                                 1, 2, 1}, since MEX = 4.
        Value of 8th index = 4.
```

A **simple solution** is to generate the series using given steps and store in an array. Finally, we return k-th element of the array.

An **efficient solution** is to use Binary Search. Observe that the middle element of our resulting sequence is n itself. Length of the sequence is $2^n - 1$, because lengths will be like

(1, 3, 7, 15....2^n -1). We use Binary search to solve this problem. As we know that the middle element of a sequence is the number of the step(from 1) performed on it.
In fact, every element in the sequence is a middle element at one or other step.
We start searching the element from step n and compare the middle index with 'k' if found we return n, else we decrease n by 1 and update our range of our search.
we repeat this step until the index is reached.

CPP

```
 // CPP program to fin k-th element after append
// and insert middle operations
#include <bits/stdc++.h>
using namespace std;
void findElement(int n, int k)
{
    int ans = n; // Middle element of the sequence
    int left = 1;

    // length of the resulting sequence.
    int right = pow(2, n) - 1;
    while (1) {
        int mid = (left + right) / 2;
        if (k == mid) {
            cout << ans << endl;
            break;
        }

        // Updating the middle element of next sequence
        ans--;

        // Moving to the left side of the middle element.
        if (k < mid)
            right = mid - 1;

        // Moving to the right side of the middle element.
        else
            left = mid + 1;
    }
}

// Driver code
int main()
{
    int n = 4, k = 8;
    findElement(n, k);
    return 0;
}
```

Python3

```
 # Python3 code to find k-th element after append
# and insert middle operations
import math

def findElement(n , k):
    ans = n      # Middle element of the sequence
    left = 1

    # length of the resulting sequence.
    right = math.pow(2, n) - 1
    while 1:
        mid = int((left + right) / 2)
        if k == mid:
            print(ans)
            break

        # Updating the middle element of next sequence
        ans-=1

        # Moving to the left side of the middle element.
        if k < mid:
            right = mid - 1

        # Moving to the right side of the middle element.
        else:
            left = mid + 1

# Driver code
n = 4
k = 8
findElement(n, k)

# This code is contributed by "Sharad_Bhardwaj".
```

PHP

```
 <?php
// PHP program to fin k-th element
// after append and insert middle
// operations

function findElement($n, $k)
{

    // Middle element of the sequence
```

```
    $ans = $n;
    $left = 1;

    // length of the resulting sequence.
    $right = pow(2, $n) - 1;
    while (1)
    {
        $mid = ($left + $right) / 2;
        if ($k ==$mid)
        {
            echo $ans ,"\n";
            break;
        }

        // Updating the middle element
        // of next sequence
        $ans--;

        // Moving to the left side of
        // the middle element.
        if ($k < $id)
            $right = $mid - 1;

        // Moving to the right side
        // of the middle element.
        else
            $left = $mid + 1;
    }
}

    // Driver code
    $n = 4;
    $k = 8;
    findElement($n, $k);

// This code is contributed by aj_36
?>
```

Output:

```
4
```

Time Complexity:O(log n)

Improved By : jit_t

Source

https://www.geeksforgeeks.org/value-k-th-index-series-formed-append-insert-mex-middle/

Chapter 261

Variants of Binary Search

Variants of Binary Search - GeeksforGeeks

Binary search is very easy right? Well, binary search can become complex when element duplication occurs in the sorted list of values. It's not always the "contains or not" we search using Binary Search, but there are 5 variants such as below:

1) Contains (True or False)
2) Index of first occurrence of a key
3) Index of last occurrence of a key
4) Index of least element greater than key
5) Index of greatest element less than key

Each of these searches, while the base logic remains same, have a minor variation in implementation and competitive coders should be aware of them. You might have seen other approaches such as this for finding first and last occurrence, where you compare adjacent element also for checking of first/last element is reached.
From a complexity perspective, it may look like an O(log n) algorithm, but it doesn't work when the comparisons itself are expensive. A problem to prove this point is linked at the end of this post, feel free to try it out.

Variant 1: Contains key (True or False)

```
Input : 2 3 3 5 5 5 6 6
Function : Contains(4)
Returns : False

Function : Contains(5)
Returns : True
```

Variant 2: First occurrence of key (index of array). This is similar to

```
std::lower_bound(...)
```

```
Input : 2 3 3 5 5 5 6 6
Function : first(3)
Returns : 1

Function : first(5)
Returns : 3

Function : first(4)
Returns : -1
```

Variant 3: Last occurrence of key (index of array)

```
Input : 2 3 3 5 5 5 6 6
Function : last(3)
Returns : 2

Function : last(5)
Returns : 5

Function : last(4)
Returns : -1
```

Variant 4: index(first occurrence) of least integer greater than key. This is similar to

```
std::upper_bound(...)
```

```
Input : 2 3 3 5 5 5 6 6
Function : leastGreater(2)
Returns : 1

Function : leastGreater(5)
Returns : 6
```

Variant 5: index(first occurrence) of greatest integer lesser than key

```
Input : 2 3 3 5 5 5 6 6
Function : leastGreater(2)
Returns : -1

Function : leastGreater(5)
Returns : 2
```

As you will see below, if you observe the clear difference between the implementations you will see that same logic is used to find different variants of binary search.

```
 #include <bits/stdc++.h>

using namespace std;

int n = 8; // array size
int a[] = { 2, 3, 3, 5, 5, 5, 6, 6 }; // Sorted array

/* Find if key is in array
 * Returns: True if key belongs to array,
 *          False if key doesn't belong to array */
bool contains(int low, int high, int key)
{
    bool ans = false;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        int midVal = a[mid];

        if (midVal < key) {

            // if mid is less than key, all elements
            // in range [low, mid] are also less
            // so we now search in [mid + 1, high]
            low = mid + 1;
        }
        else if (midVal > key) {

            // if mid is greater than key, all elements
            // in range [mid + 1, high] are also greater
            // so we now search in [low, mid - 1]
            high = mid - 1;
        }
        else if (midVal == key) {

            // comparison added just for the sake
            // of clarity if mid is equal to key, we
            // have found that key exists in array
            ans = true;
            break;
        }
    }

    return ans;
}

/* Find first occurrence index of key in array
```

```
 * Returns: an index in range [0, n-1] if key belongs
 *          to array, -1 if key doesn't belong to array
 */
int first(int low, int high, int key)
{
    int ans = -1;

    while (low <= high) {
        int mid = low + (high - low + 1) / 2;
        int midVal = a[mid];

        if (midVal < key) {

            // if mid is less than key, all elements
            // in range [low, mid] are also less
            // so we now search in [mid + 1, high]
            low = mid + 1;
        }
        else if (midVal > key) {

            // if mid is greater than key, all elements
            // in range [mid + 1, high] are also greater
            // so we now search in [low, mid - 1]
            high = mid - 1;
        }
        else if (midVal == key) {

            // if mid is equal to key, we note down
            //  the last found index then we search
            // for more in left side of mid
            // so we now search in [low, mid - 1]
            ans = mid;
            high = mid - 1;
        }
    }

    return ans;
}

/* Find last occurrence index of key in array
 * Returns: an index in range [0, n-1] if key
             belongs to array,
 *          -1 if key doesn't belong to array
 */
int last(int low, int high, int key)
{
    int ans = -1;
```

```
    while (low <= high) {
        int mid = low + (high - low + 1) / 2;
        int midVal = a[mid];

        if (midVal < key) {

            // if mid is less than key, then all elements
            // in range [low, mid - 1] are also less
            // so we now search in [mid + 1, high]
            low = mid + 1;
        }
        else if (midVal > key) {

            // if mid is greater than key, then all
            // elements in range [mid + 1, high] are
            // also greater so we now search in
            // [low, mid - 1]
            high = mid - 1;
        }
        else if (midVal == key) {

            // if mid is equal to key, we note down
            // the last found index then we search
            // for more in right side of mid
            // so we now search in [mid + 1, high]
            ans = mid;
            low = mid + 1;
        }
    }

    return ans;
}

/* Find index of first occurrence of least element
   greater than key in array
 * Returns: an index in range [0, n-1] if key is not
             the greatest element in array,
 *          -1 if key is the greatest element in array */
int leastgreater(int low, int high, int key)
{
    int ans = -1;

    while (low <= high) {
        int mid = low + (high - low + 1) / 2;
        int midVal = a[mid];

        if (midVal < key) {
```

```
            // if mid is less than key, all elements
            // in range [low, mid - 1] are <= key
            // then we search in right side of mid
            // so we now search in [mid + 1, high]
            low = mid + 1;
        }
        else if (midVal > key) {

            // if mid is greater than key, all elements
            // in range [mid + 1, high] are >= key
            // we note down the last found index, then
            // we search in left side of mid
            // so we now search in [low, mid - 1]
            ans = mid;
            high = mid - 1;
        }
        else if (midVal == key) {

            // if mid is equal to key, all elements in
            // range [low, mid] are <= key
            // so we now search in [mid + 1, high]
            low = mid + 1;
        }
    }

    return ans;
}

/* Find index of last occurrence of greatest element
   less than key in array
 * Returns: an index in range [0, n-1] if key is not
             the least element in array,
 *          -1 if key is the least element in array */
int greatestlesser(int low, int high, int key)
{
    int ans = -1;

    while (low <= high) {
        int mid = low + (high - low + 1) / 2;
        int midVal = a[mid];

        if (midVal < key) {

            // if mid is less than key, all elements
            // in range [low, mid - 1] are < key
            // we note down the last found index, then
            // we search in right side of mid
            // so we now search in [mid + 1, high]
```

```
            ans = mid;
            low = mid + 1;
        }
        else if (midVal > key) {

            // if mid is greater than key, all elements
            // in range [mid + 1, high] are > key
            // then we search in left side of mid
            // so we now search in [low, mid - 1]
            high = mid - 1;
        }
        else if (midVal == key) {

            // if mid is equal to key, all elements
            // in range [mid + 1, high] are >= key
            // then we search in left side of mid
            // so we now search in [low, mid - 1]
            high = mid - 1;
        }
    }

    return ans;
}

int main()
{
    printf("Contains\n");
    for (int i = 0; i < 10; i++)
        printf("%d %d\n", i, contains(0, n - 1, i));

    printf("First occurence of key\n");
    for (int i = 0; i < 10; i++)
        printf("%d %d\n", i, first(0, n - 1, i));

    printf("Last occurence of key\n");
    for (int i = 0; i < 10; i++)
        printf("%d %d\n", i, last(0, n - 1, i));

    printf("Least integer greater than key\n");
    for (int i = 0; i < 10; i++)
        printf("%d %d\n", i, leastgreater(0, n - 1, i));

    printf("Greatest integer lesser than key\n");
    for (int i = 0; i < 10; i++)
        printf("%d %d\n", i, greatestlesser(0, n - 1, i));

    return 0;
}
```

Output:

```
Contains
0 0
1 0
2 1
3 1
4 0
5 1
6 1
7 0
8 0
9 0
First occurence of key
0 -1
1 -1
2 0
3 1
4 -1
5 3
6 6
7 -1
8 -1
9 -1
Last occurence of key
0 -1
1 -1
2 0
3 2
4 -1
5 5
6 7
7 -1
8 -1
9 -1
Least integer greater than key
0 0
1 0
2 1
3 3
4 3
5 6
6 -1
7 -1
8 -1
9 -1
Greatest integer lesser than key
```

```
0 -1
1 -1
2 -1
3 0
4 2
5 2
6 5
7 7
8 7
9 7
```

Here is the problem I have mentioned at the beginning of the post: KCOMPRES problem in Codechef. Do try it out and feel free post your queries here.

More Binary Search Practice Problems

Source

https://www.geeksforgeeks.org/variants-of-binary-search/

Chapter 262

Ways to choose three points with distance between the most distant points <= L

Ways to choose three points with distance between the most distant points <= L - GeeksforGeeks

Given a set of n distinct points x_1, x_2, x_3... x_n all lying on the X-axis and an integer L, the task is to find the number of ways of selecting three points such that the distance between the most distant points is less than or equal to L

Note: Order is not important i.e the points {3, 2, 1} and {1, 2, 3} represent the same set of three points

Examples:

> Input : x = {1, 2, 3, 4}
> L = 3
> Output : 4
>
> **Explanation:**
> Ways to select three points such that the distance between
> the most distant points <= L are:
> 1) {1, 2, 3} Here distance between farthest points = 3 – 1 = 2 <= L
> 2) {1, 2, 4} Here distance between farthest points = 4 – 1 = 3 <= L
> 3) {1, 3, 4} Here distance between farthest points = 4 – 1 = 3 <= L
> 4) {2, 3, 4} Here distance between farthest points = 4 – 2 = 2 <= L
>
> **Thus, total number of ways = 4**

Naive Approach:
First of all, sort the array of points to generate triplets {a, b, c} such that a and c are the farthest points of the triplet and a < b < c, since all the points are distinct. We can

generate all the possible triplets and check for the condition if the distance between the two most distant points in <= L. If it holds we count this way, else we don't

C++

```
 // C++ program to count ways to choose
// triplets such that the distance
// between the farthest points <= L
#include<bits/stdc++.h>
using namespace std;

// Returns the number of triplets with
// distance between farthest points <= L
int countTripletsLessThanL(int n, int L, int* arr)
{
    // sort to get ordered triplets so that we can
    // find the distance between farthest points
    // belonging to a triplet
    sort(arr, arr + n);

    int ways = 0;

    // generate and check for all possible
    // triplets: {arr[i], arr[j], arr[k]}
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            for (int k = j + 1; k < n; k++) {

                // Since the array is sorted the
                // farthest points will be a[i]
                // and a[k];
                int mostDistantDistance = arr[k] - arr[i];
                if (mostDistantDistance <= L) {
                    ways++;
                }
            }
        }
    }

    return ways;
}

// Driver Code
int main()
{
    // set of n points on the X axis
    int arr[] = { 1, 2, 3, 4 };
```

```
    int n = sizeof(arr) / sizeof(arr[0]);
    int L = 3;
    int ans = countTripletsLessThanL(n, L, arr);
    cout << "Total Number of ways = " << ans << "\n";
    return 0;
}
```

Java

```
 // Java program to count ways to choose
// triplets such that the distance
// between the farthest points <= L
import java .io.*;
import java .util.Arrays;
class GFG {

    // Returns the number of triplets with
    // distance between farthest points <= L
    static int countTripletsLessThanL(int n, int L,
                                      int []arr)
    {

        // sort to get ordered triplets
        // so that we can find the
        // distance between farthest
        // points belonging to a triplet
        Arrays.sort(arr);

        int ways = 0;

        // generate and check for all possible
        // triplets: {arr[i], arr[j], arr[k]}
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                for (int k = j + 1; k < n; k++) {

                    // Since the array is sorted the
                    // farthest points will be a[i]
                    // and a[k];
                    int mostDistantDistance =
                                    arr[k] - arr[i];
                    if (mostDistantDistance <= L)
                    {
                        ways++;
                    }
                }
            }
        }
```

```
        return ways;
    }

    // Driver Code
    static public void main (String[] args)
    {

        // set of n points on the X axis
        int []arr = {1, 2, 3, 4};

        int n =arr.length;
        int L = 3;
        int ans = countTripletsLessThanL(n, L, arr);
        System.out.println("Total Number of ways = "
                                              + ans);
    }
}

// This code is contributed by anuj_67.
```

C#

```
 // C# program to count ways to choose
// triplets such that the distance
// between the farthest points <= L
using System;
class GFG {

// Returns the number of triplets with
// distance between farthest points <= L
static int countTripletsLessThanL(int n, int L,
                                  int []arr)
{

    // sort to get ordered triplets
    // so that we can find the
    // distance between farthest
    // points belonging to a triplet
    Array.Sort(arr);

    int ways = 0;

    // generate and check for all possible
    // triplets: {arr[i], arr[j], arr[k]}
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            for (int k = j + 1; k < n; k++) {
```

```
                // Since the array is sorted the
                // farthest points will be a[i]
                // and a[k];
                int mostDistantDistance = arr[k] - arr[i];
                if (mostDistantDistance <= L)
                {
                    ways++;
                }
            }
        }
    }

    return ways;
}

    // Driver Code
    static public void Main ()
    {

        // set of n points on the X axis
        int []arr = {1, 2, 3, 4};

        int n =arr.Length;
        int L = 3;
        int ans = countTripletsLessThanL(n, L, arr);
        Console.WriteLine("Total Number of ways = " + ans);
    }
}

// This code is contributed by anuj_67.
```

Output

```
Total Number of ways = 4
```

Time Complexity: $\mathbf{O(n^3)}$ for generating all possible triplets.

Efficient Approach:

- This problem can be solved by using Binary search.
- First of all, sort the array.
- Now, for each element of the array we find the number of elements which are greater than it(by maintaining a sorted order of points) and lie in the range ($x_i + 1$, $x_i + L$) both inclusive (Note that here all points are distinct so we need consider the elements equal to x_i itself).

- Doing so we find all such points where the distance between the farthest points will always be less than or equal to L.
- Now let's say for the i^{th} point, we have M such points which are less than or equal to x_i + L, then the number of ways we can select 2 points from M such points is simply M * (M – 1) / 2

```
 // C++ program to count ways to choose
// triplets such that the distance between
// the farthest points <= L */
#include<bits/stdc++.h>
using namespace std;

// Returns the number of triplets with the
// distance between farthest points <= L
int countTripletsLessThanL(int n, int L, int* arr)
{
    // sort the array
    sort(arr, arr + n);

    int ways = 0;
    for (int i = 0; i < n; i++) {

        // find index of element greater than arr[i] + L
        int indexGreater = upper_bound(arr, arr + n,
                                       arr[i] + L) - arr;

        // find Number of elements between the ith
        // index and indexGreater since the Numbers
        // are sorted and the elements are distinct
        // from the points btw these indices represent
        // points within range (a[i] + 1 and a[i] + L)
        // both inclusive

        int numberOfElements = indexGreater - (i + 1);

        // if there are at least two elements in between
        // i and indexGreater find the Number of ways
        // to select two points out of these

        if (numberOfElements >= 2) {
            ways += (numberOfElements
                        * (numberOfElements - 1) / 2);
        }
    }

    return ways;
}
```

```
// Driver Code
int main()
{
    // set of n points on the X axis
    int arr[] = { 1, 2, 3, 4 };

    int n = sizeof(arr) / sizeof(arr[0]);
    int L = 4;

    int ans = countTripletsLessThanL(n, L, arr);

    cout << "Total Number of ways = " << ans << "\n";

    return 0;
}
```

Output

```
Total Number of ways = 4
```

Time Complexity:**O(NlogN)** where N is the number of points.

Improved By : vt_m

Source

https://www.geeksforgeeks.org/ways-choose-three-points-distance-distant-points-1/

Chapter 263

Why is Binary Search preferred over Ternary Search?

Why is Binary Search preferred over Ternary Search? - GeeksforGeeks

The following is a simple recursive **Binary Search** function in C++ taken from here.

```
 // A recursive binary search function. It returns location of x in
// given array arr[l..r] is present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
   if (r >= l)
   {
        int mid = l + (r - l)/2;

        // If the element is present at the middle itself
        if (arr[mid] == x)  return mid;

        // If element is smaller than mid, then it can only be present
        // in left subarray
        if (arr[mid] > x) return binarySearch(arr, l, mid-1, x);

        // Else the element can only be present in right subarray
        return binarySearch(arr, mid+1, r, x);
   }

   // We reach here when element is not present in array
   return -1;
}
```

The following is a simple recursive **Ternary Search** function :

C

```
 // A recursive ternary search function. It returns location of x in
// given array arr[l..r] is present, otherwise -1
int ternarySearch(int arr[], int l, int r, int x)
{
   if (r >= l)
   {
        int mid1 = l + (r - l)/3;
        int mid2 = mid1 + (r - l)/3;

        // If x is present at the mid1
        if (arr[mid1] == x)  return mid1;

        // If x is present at the mid2
        if (arr[mid2] == x)  return mid2;

        // If x is present in left one-third
        if (arr[mid1] > x) return ternarySearch(arr, l, mid1-1, x);

        // If x is present in right one-third
        if (arr[mid2] < x) return ternarySearch(arr, mid2+1, r, x);

        // If x is present in middle one-third
        return ternarySearch(arr, mid1+1, mid2-1, x);
   }
   // We reach here when element is not present in array
   return -1;
}
```

PHP

```
 <?php
// A recursive ternary search function.
// It returns location of x in
// given array arr[l..r] is
// present, otherwise -1
function ternarySearch($arr, $l,
                       $r, $x)
{
    if ($r >= $l)
    {
        $mid1 = $l + ($r - $l) / 3;
        $mid2 = $mid1 + ($r - l) / 3;

        // If x is present at the mid1
        if ($arr[mid1] == $x)
            return $mid1;

        // If x is present
```

```
        // at the mid2
        if ($arr[$mid2] == $x)
            return $mid2;

        // If x is present in
        // left one-third
        if ($arr[$mid1] > $x)
            return ternarySearch($arr, $l,
                                 $mid1 - 1, $x);

        // If x is present in right one-third
        if ($arr[$mid2] < $x)
            return ternarySearch($arr, $mid2 + 1,
                                       $r, $x);

        // If x is present in
        // middle one-third
        return ternarySearch($arr, $mid1 + 1,
                             $mid2 - 1, $x);
}

// We reach here when element
// is not present in array
return -1;
}

// This code is contributed by anuj_67
?>
```

Which of the above two does less comparisons in worst case?
From the first look, it seems the ternary search does less number of comparisons as it makes Log_3n recursive calls, but binary search makes Log_2n recursive calls. Let us take a closer look.
The following is recursive formula for counting comparisons in worst case of Binary Search.

```
T(n) = T(n/2) + 2,  T(1) = 1
```

The following is recursive formula for counting comparisons in worst case of Ternary Search.

```
T(n) = T(n/3) + 4, T(1) = 1
```

In binary search, there are $2Log_2n + 1$ comparisons in worst case. In ternary search, there are $4Log_3n + 1$ comparisons in worst case.

```
Time Complexity for Binary search = 2clog2n + O(1)
Time Complexity for Ternary search = 4clog3n + O(1)
```

Therefore, the comparison of Ternary and Binary Searches boils down the comparison of expressions $2Log_3n$ and Log_2n . The value of $2Log_3n$ can be written as (2 / $Log_2 3$) * Log_2n . Since the value of (2 / $Log_2 3$) is more than one, Ternary Search does more comparisons than Binary Search in worst case.

Exercise:
Why Merge Sort divides input array in two halves, why not in three or more parts?

This article is contributed by **Anmol**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : vt_m

Source

https://www.geeksforgeeks.org/binary-search-preferred-ternary-search/

Chapter 264

XOR of numbers that appeared even number of times in given Range

XOR of numbers that appeared even number of times in given Range - GeeksforGeeks

Given an array of numbers of size N and Q queries. Each query or a range can be represented by L (LeftIndex) and R(RightIndex). Find the XOR-sum of the numbers that appeared even number of times in the given range.

Prerequisite : Queries for number of distinct numbers in given range. | Segment Tree for range query

Examples :

```
Input : arr[] = { 1, 2, 1, 3, 3, 2, 3 }
        Q = 5
        L = 3,  R = 6
        L = 3,  R = 4
        L = 0,  R = 2
        L = 0,  R = 6
        L = 0,  R = 4
Output : 0
         3
         1
         3
         2
```

Explanation of above example:
In Query 1, there are no numbers which appeared even number of times.
Hence the XOR-sum is 0.

In Query 2, {3} appeared even number of times. XOR-sum is 3.
In Query 3, {1} appeared even number of times. XOR-sum is 1.
In Query 4, {1, 2} appeared even number of times. XOR-sum is 1 xor 2 = 3.
In Query 5, {1, 3} appeared even number of times. XOR-sum is 1 xor 3 = 2.

Segment Trees or Binary Indexed Trees can be used to solve this problem efficiently.

Approach :
Firstly, it is easy to note that the answer for the query is the XOR-sum of **all** elements in the query range xor-ed with XOR-sum of **distinct** elements in the query range (since taking XOR of an element with itself results into a null value). Find the XOR-sum of all numbers in query range using prefix XOR-sums.
To find the XOR-sum of distinct elements in range : Number of distinct elements in a subarray of given range.
Now, returning back to our main problem, just change the assignment BIT[i] = 1 to BIT[i] = arr_i and count the XOR-sum instead of sum.

Below is the implementation using Binary Indexed Trees in CPP

```
 // CPP Program to Find the XOR-sum
// of elements that appeared even
// number of times within a range
#include <bits/stdc++.h>
using namespace std;

/* structure to store queries
   L --> Left Bound of Query
   R --> Right Bound of Query
   idx --> Query Number */
struct que {
    int L, R, idx;
};

// cmp function to sort queries
// according to R
bool cmp(que a, que b)
{
    if (a.R != b.R)
        return a.R < b.R;
    else
        return a.L < b.L;
}

/*  N  --> Number of elements present in
    input array. BIT[0..N] --> Array that
    represents Binary Indexed Tree*/

// Returns XOR-sum of arr[0..index]. This
// function assumes that the array is
```

```
// preprocessed and partial sums of array
// elements are stored in BIT[].
int getSum(int BIT[], int index)
{
    // Iniialize result
    int xorSum = 0;

    // index in BITree[] is 1 more than
    // the index in arr[]
    index = index + 1;

    // Traverse ancestors of BIT[index]
    while (index > 0)
    {
        // Take XOR of current element
        // of BIT to xorSum
        xorSum ^= BIT[index];

        // Move index to parent node
        // in getSum View
        index -= index & (-index);
    }
    return xorSum;
}

// Updates a node in Binary Index Tree
// (BIT) at given index in BIT.  The
// given value 'val' is xored to BIT[i]
// and all of its ancestors in tree.
void updateBIT(int BIT[], int N,
               int index, int val)
{
    // index in BITree[] is 1 more than
    // the index in arr[]
    index = index + 1;

    // Traverse all ancestors and
    // take xor with 'val'
    while (index <= N)
    {
        // Take xor with 'val' to
        // current node of BIT
        BIT[index] ^= val;

        // Update index to that of
        // parent in update View
        index += index & (-index);
    }
```

```
}

// Constructs and returns a Binary Indexed
// Tree for given array of size N.
int* constructBITree(int arr[], int N)
{
    // Create and initialize BITree[] as 0
    int* BIT = new int[N + 1];

    for (int i = 1; i <= N; i++)
        BIT[i] = 0;

    return BIT;
}

// Function to answer the Queries
void answeringQueries(int arr[], int N,
        que queries[], int Q, int BIT[])
{
    // Creating an array to calculate
    // prefix XOR sums
    int* prefixXOR = new int[N + 1];

    // map for coordinate compression
    // as numbers can be very large but we
    // have limited space
    map<int, int> mp;

    for (int i = 0; i < N; i++) {

        // If A[i] has not appeared yet
        if (!mp[arr[i]])
            mp[arr[i]] = i;

        // calculate prefixXOR sums
        if (i == 0)
            prefixXOR[i] = arr[i];
        else
            prefixXOR[i] =
                prefixXOR[i - 1] ^ arr[i];
    }

    // Creating an array to store the
    // last occurence of arr[i]
    int lastOcc[1000001];
    memset(lastOcc, -1, sizeof(lastOcc));

    // sort the queries according to comparator
```

```
    sort(queries, queries + Q, cmp);

    // answer for each query
    int res[Q];

    // Query Counter
    int j = 0;

    for (int i = 0; i < Q; i++)
    {
        while (j <= queries[i].R)
        {
            // If last visit is not -1 update
            // arr[j] to set null by taking
            // xor with itself at the idx
            // equal lastOcc[mp[arr[j]]]
            if (lastOcc[mp[arr[j]]] != -1)
                updateBIT(BIT, N,
                      lastOcc[mp[arr[j]]], arr[j]);

            // Setting lastOcc[mp[arr[j]]] as j and
            // updating the BIT array accordingly
            updateBIT(BIT, N, j, arr[j]);
            lastOcc[mp[arr[j]]] = j;
            j++;
        }

        // get the XOR-sum of all elements within
        // range using precomputed prefix XORsums
        int allXOR = prefixXOR[queries[i].R] ^
                     prefixXOR[queries[i].L - 1];

        // get the XOR-sum of distinct elements
        // within range using BIT query function
        int distinctXOR = getSum(BIT, queries[i].R) ^
                          getSum(BIT, queries[i].L - 1);

        // store the final answer at the numbered query
        res[queries[i].idx] = allXOR ^ distinctXOR;
    }

    // Output the result
    for (int i = 0; i < Q; i++)
        cout << res[i] << endl;
}

// Driver program to test above functions
int main()
```

```
{
    int arr[] = { 1, 2, 1, 3, 3, 2, 3 };
    int N = sizeof(arr) / sizeof(arr[0]);

    int* BIT = constructBITree(arr, N);

    // structure of array for queries
    que queries[5];

    // Intializing values (L, R, idx) to queries
    queries[0].L = 3;
    queries[0].R = 6, queries[0].idx = 0;
    queries[1].L = 3;
    queries[1].R = 4, queries[1].idx = 1;
    queries[2].L = 0;
    queries[2].R = 2, queries[2].idx = 2;
    queries[3].L = 0;
    queries[3].R = 6, queries[3].idx = 3;
    queries[4].L = 0;
    queries[4].R = 4, queries[4].idx = 4;

    int Q = sizeof(queries) / sizeof(queries[0]);

    // answer Queries
    answeringQueries(arr, N, queries, Q, BIT);

    return 0;
}
```

Output:

```
0
3
1
3
2
```

Time Complexity: O(Q * Log(N)), where N is the size of array, Q is the total number of queries.

Source

https://www.geeksforgeeks.org/xor-numbers-appeared-even-number-times-given-range/

Chapter 265

k largest(or smallest) elements in an array | added Min Heap method

k largest(or smallest) elements in an array | added Min Heap method - GeeksforGeeks

Question: Write an efficient program for printing k largest elements in an array. Elements in array can be in any order.

For example, if given array is [1, 23, 12, 9, 30, 2, 50] and you are asked for the largest 3 elements i.e., k = 3 then your program should print 50, 30 and 23.

Method 1 (Use Bubble k times)
Thanks to Shailendra for suggesting this approach.
1) Modify Bubble Sort to run the outer loop at most k times.
2) Print the last k elements of the array obtained in step 1.

Time Complexity: O(nk)

Like Bubble sort, other sorting algorithms like Selection Sort can also be modified to get the k largest elements.

Method 2 (Use temporary array)
K largest elements from arr[0..n-1]

1) Store the first k elements in a temporary array temp[0..k-1].
2) Find the smallest element in temp[], let the smallest element be *min*.
3) For each element *x* in arr[k] to arr[n-1]
If *x* is greater than the min then remove *min* from temp[] and insert *x*.
4) Print final k elements of *temp[]*

Time Complexity: O((n-k)*k). If we want the output sorted then O((n-k)*k + klogk)

Thanks to nesamani1822 for suggesting this method.

Method 3(Use Sorting)
1) Sort the elements in descending order in O(nLogn)
2) Print the first k numbers of the sorted array O(k).
Following is the implementation of above.

C++

```
 // C++ code for k largest elements in an array
#include<bits/stdc++.h>
using namespace std;

void kLargest(int arr[], int n, int k)
{
    // Sort the given array arr in reverse
    // order.
    sort(arr, arr+n, greater<int>());

    // Print the first kth largest elements
    for (int i = 0; i < k; i++)
        cout << arr[i] << " ";
}

// driver program
int main()
{
    int arr[] = {1, 23, 12, 9, 30, 2, 50};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    kLargest(arr, n, k);
}

// This article is contributed by Chhavi
```

Java

```
 // Java code for k largest elements in an array
import java.util.Arrays;
import java.util.Collections;

class GFG
{
    public static void kLargest(Integer [] arr, int k)
    {
        // Sort the given array arr in reverse order
        // This method doesn't work with primitive data
        // types. So, instead of int, Integer type
        // array will be used
```

```
        Arrays.sort(arr, Collections.reverseOrder());


    // Print the first kth largest elements
    for (int i = 0; i < k; i++)
    System.out.print(arr[i] + " ");
    }


    public static void main(String[] args)
    {
        Integer arr[] = new Integer[]{1, 23, 12, 9,
                                      30, 2, 50};
        int k = 3;
        kLargest(arr,k);
    }
}
// This code is contributed by Kamal Rawal
```

Python

```
 ''' Python3 code for k largest elements in an array'''

def kLargest(arr, k):
    # Sort the given array arr in reverse
    # order.
    arr.sort(reverse=True)
    #Print the first kth largest elements
    for i in range(k):
        print (arr[i],end=" ")

# Driver program
arr = [1, 23, 12, 9, 30, 2, 50]
#n = len(arr)
k = 3
kLargest(arr, k)

# This code is contributed by shreyanshi_arun.
```

Output:

```
50 30 23
```

Time complexity: O(nlogn)

Method 4 (Use Max Heap)
1) Build a Max Heap tree in O(n)
2) Use Extract Max k times to get k maximum elements from the Max Heap O(klogn)

Time complexity: O(n + klogn)

Method 5(Use Oder Statistics)
1) Use order statistic algorithm to find the kth largest element. Please see the topic selection in worst-case linear time O(n)
2) Use QuickSort Partition algorithm to partition around the kth largest number O(n).
3) Sort the k-1 elements (elements greater than the kth largest element) O(kLogk). This step is needed only if sorted output is required.

Time complexity: O(n) if we don't need the sorted output, otherwise O(n+kLogk)

Thanks to Shilpi for suggesting the first two approaches.

Method 6 (Use Min Heap)
This method is mainly an optimization of method 1. Instead of using temp[] array, use Min Heap.

1) Build a Min Heap MH of the first k elements (arr[0] to arr[k-1]) of the given array. O(k)

2) For each element, after the kth element (arr[k] to arr[n-1]), compare it with root of MH.
……a) If the element is greater than the root then make it root and call heapifyfor MH
……b) Else ignore it.
// The step 2 is O((n-k)*logk)

3) Finally, MH has k largest elements and root of the MH is the kth largest element.

Time Complexity: O(k + (n-k)Logk) without sorted output. If sorted output is needed then O(k + (n-k)Logk + kLogk)

All of the above methods can also be used to find the kth largest (or smallest) element.

Please write comments if you find any of the above explanations/algorithms incorrect, or find better ways to solve the same problem.

References:
http://en.wikipedia.org/wiki/Selection_algorithm

Source

https://www.geeksforgeeks.org/k-largestor-smallest-elements-in-an-array/

Chapter 266

k-th missing element in increasing sequence which is not present in a given sequence

k-th missing element in increasing sequence which is not present in a given sequence - GeeksforGeeks

Given two sequences, one is increasing sequence **a**[] and another a normal sequence **b**[], find the K-th missing element in the increasing sequence which is not present in the given sequence. If no k-th missing element is there output -1

Examples:

```
Input:  a[] = {0, 2, 4, 6, 8, 10, 12, 14, 15};
        b[] = {4, 10, 6, 8, 12};
          k = 3
Output: 14
Explanation : The numbers from increasing sequence that
are not present in the given sequence are 0, 2, 14, 15.
The 3rd missing number is 14.
```

n1 Number of elements on increasing sequence a[].
n2 Number of elements in given sequence b[].

A **naive approach** is to iterate for every element in the increasing sequence and check if it is present in the given sequence or not, and keep a counter of not present elements, and print the k-th non present element. This will not be efficient enough as it has two nested for loops which will take O(n2).
Time complexity: O(n1 * n2)
Auxiliary space: O(1)

An **efficient approach** is to use hashing. We store all elements of given sequence in a hash table. Then we iterate through all elements of increasing sequence. For every element, we search it in the hash table. If element is present in not hash table, then we increment count of missing elements. If count becomes k, we return the missing element.

Below is the C++ implementation of the above approach

```
 // C++ program to find the k-th missing element
// in a given sequence
#include <bits/stdc++.h>
using namespace std;

// Returns k-th missing element. It returns -1 if
// no k is more than number of missing elements.
int find(int a[], int b[], int k, int n1, int n2)
{
    // Insert all elements of givens sequence b[].
    unordered_set<int> s;
    for (int i = 0; i < n2; i++)
        s.insert(b[i]);

    // Traverse through increasing sequence and
    // keep track of count of missing numbers.
    int missing = 0;
    for (int i = 0; i < n1; i++) {
        if (s.find(a[i]) == s.end())
            missing++;
        if (missing == k)
            return a[i];
    }

    return -1;
}

// driver program to test the above function
int main()
{
    int a[] = { 0, 2, 4, 6, 8, 10, 12, 14, 15 };
    int b[] = { 4, 10, 6, 8, 12 };
    int n1 = sizeof(a) / sizeof(a[0]);
    int n2 = sizeof(b) / sizeof(b[0]);

    int k = 3;
    cout << find(a, b, k, n1, n2);
    return 0;
}
```

Output:

```
14
```

Time complexity: O(n1 + n2)
Auxiliary Space: O(n2)

Source

https://www.geeksforgeeks.org/k-th-missing-element-increasing-sequence-not-present-given-sequence/

Chapter 267

k-th missing element in sorted array

k-th missing element in sorted array - GeeksforGeeks

Given an increasing sequence **a**[], we need to find the K-th missing contiguous element in the increasing sequence which is not present in the sequence. If no k-th missing element is there output -1.

Examples :

```
Input : a[] = {2, 3, 5, 9, 10};
        k = 1;
Output : 4
Explanation: Missing Element in the increasing
sequence are {4, 6, 7, 8}. So k-th missing element
is 4

Input : a[] = {2, 3, 5, 9, 10, 11, 12};
        k = 4;
Output : 8
Explanation: missing element in the increasing
sequence are {4, 6, 7, 8}  so k-th missing
element is 8
```

Approach: Start iterating over the array elements, and for every element check if next element is consecutive or not, if not, then take difference between these two, and check if difference is greater than or equal to given k, then calculate ans = a[i] + count, else iterate for next element.

C++

```
 #include <bits/stdc++.h>
using namespace std;

// Function to find k-th
// missing element
int missingK(int a[], int k,
             int n)
{
    int difference = 0,
        ans = 0, count = k;
    bool flag = 0;

    // interating over the array
    for(int i = 0 ; i < n - 1; i++)
    {
        difference = 0;

        // check if i-th and
        // (i + 1)-th element
        // are not consecutive
        if ((a[i] + 1) != a[i + 1])
        {

            // save their difference
            difference +=
                (a[i + 1] - a[i]) - 1;

            // check for difference
            // and given k
            if (difference >= count)
                {
                    ans = a[i] + count;
                    flag = 1;
                    break;
                }
            else
                count -= difference;
        }
    }

    // if found
    if(flag)
        return ans;
    else
        return  -1;
}

// Driver code
```

```
int main()
{
    // Input array
    int a[] = {1, 5, 11, 19};

    // k-th missing element
    // to be found in the array
    int k = 11;
    int n = sizeof(a) / sizeof(a[0]);

    // calling function to
    // find missing element
    int missing = missingK(a, k, n);

    cout << missing << endl;

    return 0;
}
```

Java

```
 // Java program to check for
// even or odd
import java.io.*;
import java.util.*;

public class GFG {

    // Function to find k-th
    // missing element
    static int missingK(int []a, int k,
                                 int n)
    {
        int difference = 0,
            ans = 0, count = k;
        boolean flag = false;

        // interating over the array
        for(int i = 0 ; i < n - 1; i++)
        {
            difference = 0;

            // check if i-th and
            // (i + 1)-th element
            // are not consecutive
            if ((a[i] + 1) != a[i + 1])
            {
```

```
                // save their difference
                difference +=
                    (a[i + 1] - a[i]) - 1;

                // check for difference
                // and given k
                if (difference >= count)
                    {
                        ans = a[i] + count;
                        flag = true;
                        break;
                    }
                else
                    count -= difference;
            }
        }

        // if found
        if(flag)
            return ans;
        else
            return -1;
    }

    // Driver code
    public static void main(String args[])
    {

        // Input array
        int []a = {1, 5, 11, 19};

        // k-th missing element
        // to be found in the array
        int k = 11;
        int n = a.length;

        // calling function to
        // find missing element
        int missing = missingK(a, k, n);

        System.out.print(missing);
    }

}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

Python3

```
 # Function to find k-th
# missing element
def missingK(a, k, n) :

    difference = 0
    ans = 0
    count = k
    flag = 0

    # interating over the array
    for i in range (0, n-1) :
        difference = 0

        # check if i-th and
        # (i + 1)-th element
        # are not consecutive
        if ((a[i] + 1) != a[i + 1]) :


            # save their difference
            difference += (a[i + 1] - a[i]) - 1

            # check for difference
            # and given k
            if (difference >= count) :
                    ans = a[i] + count
                    flag = 1
                    break
            else :
                count -= difference

    # if found
    if(flag) :
        return ans
    else :
        return -1

# Driver code
# Input array
a = [1, 5, 11, 19]

# k-th missing element
# to be found in the array
k = 11
n = len(a)
```

```
# calling function to
# find missing element
missing = missingK(a, k, n)

print(missing)

# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
 // C# program to check for
// even or odd
using System;
using System.Collections.Generic;

class GFG {

    // Function to find k-th
    // missing element
    static int missingK(int []a, int k,
                                int n)
    {
        int difference = 0,
            ans = 0, count = k;
        bool flag = false;

        // interating over the array
        for(int i = 0 ; i < n - 1; i++)
        {
            difference = 0;

            // check if i-th and
            // (i + 1)-th element
            // are not consecutive
            if ((a[i] + 1) != a[i + 1])
            {

                // save their difference
                difference +=
                    (a[i + 1] - a[i]) - 1;

                // check for difference
                // and given k
                if (difference >= count)
                    {
                        ans = a[i] + count;
                        flag = true;
```

```
                        break;
                    }
                else
                    count -= difference;
            }
        }

        // if found
        if(flag)
            return ans;
        else
            return -1;
    }

    // Driver code
    public static void Main()
    {

        // Input array
        int []a = {1, 5, 11, 19};

        // k-th missing element
        // to be found in the array
        int k = 11;
        int n = a.Length;

        // calling function to
        // find missing element
        int missing = missingK(a, k, n);

        Console.Write(missing);
    }

}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

PHP

```
 <?php
// Function to find k-th
// missing element
function missingK(&$a, $k, $n)
{
    $difference = 0;
    $ans = 0;
    $count = $k;
```

```
    $flag = 0;

    // interating over the array
    for($i = 0 ; $i < $n - 1; $i++)
    {
        $difference = 0;

        // check if i-th and
        // (i + 1)-th element
        // are not consecutive
        if (($a[$i] + 1) != $a[$i + 1])
        {

            // save their difference
            $difference += ($a[$i + 1] -
                           $a[$i]) - 1;

            // check for difference
            // and given k
            if ($difference >= $count)
                {
                    $ans = $a[$i] + $count;
                    $flag = 1;
                    break;
                }
            else
                $count -= $difference;
        }
    }

    // if found
    if($flag)
        return $ans;
    else
        return -1;
}

// Driver Code

// Input array
$a = array(1, 5, 11, 19);

// k-th missing element
// to be found in the array
$k = 11;
$n = count($a);

// calling function to
```

```
// find missing element
$missing = missingK($a, $k, $n);

echo $missing;

// This code is contributed by Manish Shaw
// (manishshaw1)
?>
```

Output :

```
14
```

Time Complexity :O(n), where n is the number of elements in the array.

Improved By : manishshaw1

Source

https://www.geeksforgeeks.org/k-th-missing-element-in-sorted-array/