DSA NOTES

ARRAY

~ Himanshi Sehgal

1. Subarray with given sum
2. Count the triplets
3. Kadane's Algorithm
4. Missing number in array
5. Merge two sorted arrays
6. Rearrange array alternatively
7. Number of pairs
8. Inversion of Array
9. Sort an array of 0s, 1s and 2s
10. Equilibrium point
11. Leaders in an array
12. Minimum Platforms
13. Reverse array in groups
14. K'th smallest element
15. Trapping Rain Water
16. Pythagorean Triplet
17. Chocolate Distribution Problem
18. Stock buy and sell
19. Element with left side smaller and right side greater
20. Convert array into Zig-Zag fashion
21. Last Index of 1
22. Spirally traversing a matrix
23. Largest Number formed from an Array

IMP Questions

→ $\frac{n*(n+1)}{2}$

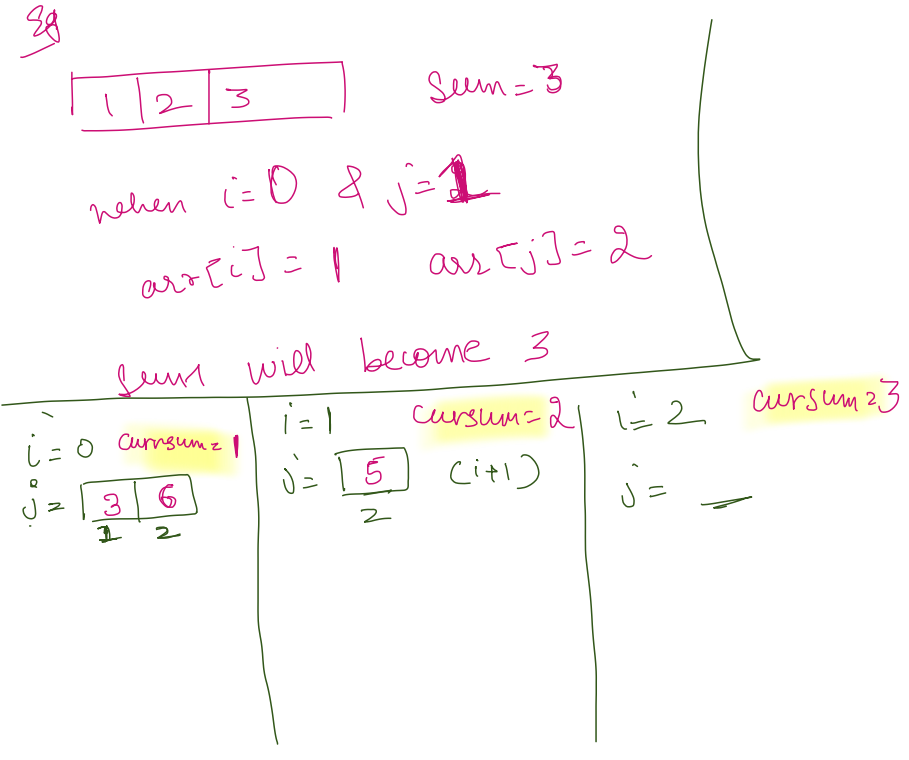
① Subarray with given Sum (All element +ve)

```
Input: arr[] = {1, 4, 20, 3, 10, 5}, sum = 33
Output: Sum found between indexes 2 and 4
Sum of elements between indices
2 and 4 is 20 + 3 + 10 = 33

Input: arr[] = {1, 4, 0, 0, 3, 10, 5}, sum = 7
Output: Sum found between indexes 1 and 4
Sum of elements between indices
1 and 4 is 4 + 0 + 0 + 3 = 7

Input: arr[] = {1, 4}, sum = 0
Output: No subarray found
There is no subarray with 0 sum
```

Simple Approach: A simple solution is to consider all subarrays one by one and check the sum of every subarray. Following program implements the simple solution. Run two loops: the outer loop picks a starting point I and the inner loop tries all subarrays starting from i.

Algorithm:

1. Traverse the array from start to end.
2. From every index start another loop from *i* to the end of array to get all subarray starting from i, keep a variable sum to calculate the sum.
3. For every index in inner loop update *sum = sum + array[j]*
4. If the sum is equal to the given sum then print the subarray.




```cpp
#include <bits/stdc++.h>
using namespace std;

/* Returns true if the there is a subarray
of arr[] with sum equal to 'sum' otherwise
returns false. Also, prints the result */
int subArraySum(int arr[], int n, int sum)
{
    int curr_sum, i, j;

    // Pick a starting point
    for (i = 0; i < n; i++) {
        curr_sum = arr[i];

        // try all subarrays starting with 'i'
        for (j = i + 1; j <= n; j++) {
            if (curr_sum == sum) {
                cout << "Sum found between indexes "
                     << i << " and " << j - 1;
                return 1;
            }
            if (curr_sum > sum || j == n)
                break;
            curr_sum = curr_sum + arr[j];
        }
    }

    cout << "No subarray found";
    return 0;
}
```

~ Himanshi Sehgal

Efficient Approach: There is an idea if all the elements of the array are positive. If a subarray has sum greater than the given sum then there is no possibility that adding elements to the current subarray the sum will be *x* (given sum). Idea is to use a similar approach to a sliding window. Start with an empty subarray, add elements to the subarray until the sum is less than *x*. If the sum is greater than *x*, remove elements from the start of the current subarray.

Algorithm:

1. Create three variables, *l=0, sum = 0*
2. Traverse the array from start to end.
3. Update the variable sum by adding current element, *sum = sum + array[i]*
4. If the sum is greater than the given sum, update the variable sum as *sum = sum - array[l]*, and update l as, l++.
5. If the sum is equal to given sum, print the subarray and break the loop.

```
returns false. Also, prints the result */
int subArraySum(int arr[], int n, int sum)
{
    /* Initialize curr_sum as value of
    first element and starting point as 0 */
    int curr_sum = arr[0], start = 0, i;

    /* Add elements one by one to curr_sum and
    if the curr_sum exceeds the sum,
    then remove starting element */
    for (i = 1; i <= n; i++) {
        // If curr_sum exceeds the sum,
        // then remove the starting elements
        while (curr_sum > sum && start < i - 1) {
            curr_sum = curr_sum - arr[start];
            start++;
        }

        // If curr_sum becomes equal to sum,
        // then return true
        if (curr_sum == sum) {
            cout << "Sum found between indexes "
                 << start << " and " << i - 1;
            return 1;
        }

        // Add this element to curr_sum
        if (i < n)
            curr_sum = curr_sum + arr[i];
    }

    // If we reach here, then no subarray
    cout << "No subarray found";
    return 0;
```

age currsum ki value jda loggi hai to left vale ko increment krdo because we have only positive numbers

~ Himanshi Sehgal

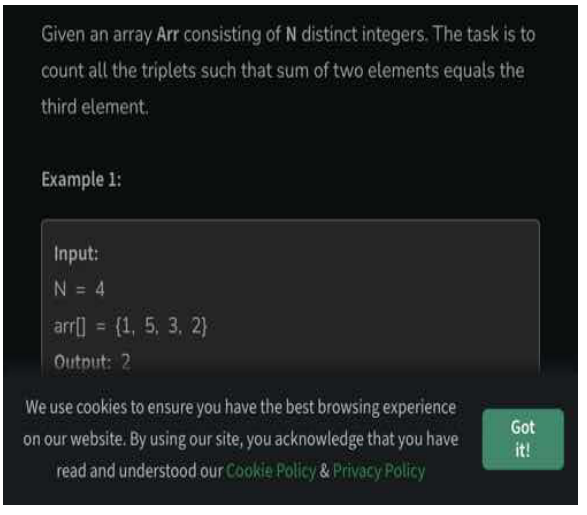
Count the triplets

Given an array of *distinct positive integers* **arr[]** of length **N**, the task is to count all the triplets such that sum of two elements equals the third element.

Examples:

Input: *arr[] = {1, 5, 3, 2}*
Output: *2*
Explanation:
In the given array, there are two such triplets such that sum of the two numbers is equal to the third number, those are -
(1, 2, 3), (3, 2, 5)
Input: *arr[] = {3, 2, 7}*
Output: *0*
Explanation:
In the given array there are no such triplets such that sum of two n umbers is equal to the third number.

hint →

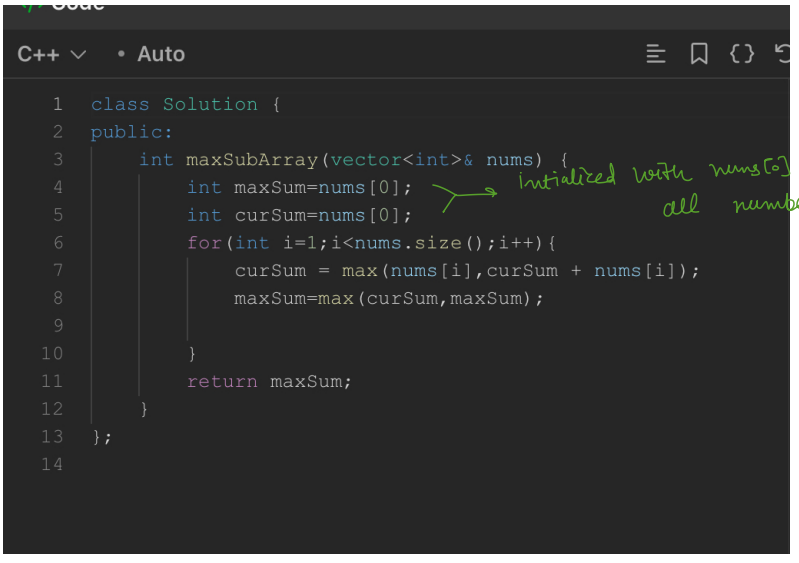


~ Himanshi Sehgal

Kadanes algo for finding max subarray sum

Input:
N = 5
Arr[] = {1,2,3,-2,5}
Output:
9
Explanation:
Max subarray sum is 9
of elements (1, 2, 3, -2, 5) which

is a contiguous subarray.



initialized with nums[0] not 0 as because all numbers can be negative aswell

Time Comp - O(n)

~ Himanshi Sehgal

Number of pairs

Number of pairs

Medium Accuracy: 47.19%

Submissions: 23441 Points: 4

Input:
M = 3, X[] = [2 1 6]
N = 2, Y[] = [1 5]
Output: 3
Explanation:
The pairs which follow $x^y > y^x$ are

as such: $2^1 > 1^2$, $2^5 > 5^2$ and $6^1 > 1^6$.

Efficient Solution:

The problem can be solved in **O(nLogn + mLogn)** time. The trick here is if **y > x** then **x^y > y^x** with some exceptions.

Following are simple steps based on this trick.

- Sort array Y[].
- For every x in X[], find the index idx of the smallest number greater than x (also called ceil of x) in Y[] using **binary search, or** we can use the inbuilt function upper_bound() in algorithm library.
- All the numbers after idx satisfy the relation so just add (n-idx) to the count.

- If x = 0, then the count of pairs for this x is 0.
- If x = 1, then the count of pairs for this x is equal to count of 0s in Y[].
- x smaller than y means x^y is greater than y^x.
  1. x = 2, y = 3 or 4
  2. x = 3, y = 2

Note that the case where x = 4 and y = 2 is not there

Following diagram shows all exceptions in tabular form. The value 1 indicates that the corresponding (x, y) form a valid pair.

	0	1	2	3	4
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	1
4	1	1	0	0	0

In the following implementation, we pre-process the Y array and count 0, 1, 2, 3 and 4 in it, so that we can handle all exceptions in constant time. The array NoOfY[] is used to store the counts.

```
int count(int x, int Y[], int n, int NoOfY[])
{
    // If x is 0, then there cannot be any value in Y su
    // that x^Y[i] > Y[i]^x
    if (x == 0)
        return 0;

    // If x is 1, then the number of pais is equal to nu
    // of zeroes in Y[]
    if (x == 1)
        return NoOfY[0];

    // Find number of elements in Y[] with values greate
    // than x upper_bound() gets address of first greate
    // element in Y[0..n-1]
    int* idx = upper_bound(Y, Y + n, x);
    int ans = (Y + n) - idx;

    // If we have reached here, then x must be greater t
    // 1, increase number of pairs for y=0 and y=1
    ans += (NoOfY[0] + NoOfY[1]);

    // Decrease number of pairs for x=2 and (y=4 or y=3)
    if (x == 2)
        ans -= (NoOfY[3] + NoOfY[4]);

    // Increase number of pairs for x=3 and y=2
    if (x == 3)
        ans += NoOfY[2];

    return ans;
}

// Function to return count of pairs (x, y) such that
// x belongs to X[], y belongs to Y[] and x^y > y^x
int countPairs(int X[], int Y[], int m, int n)
{
    // To store counts of 0, 1, 2, 3 and 4 in array Y
    int NoOfY[5] = { 0 };
    for (int i = 0; i < n; i++)
        if (Y[i] < 5)
            NoOfY[Y[i]]++;

    // Sort Y[] so that we can do binary search in it
    sort(Y, Y + n);

    int total_pairs = 0; // Initialize result

    // Take every element of X and count pairs with it
    for (int i = 0; i < m; i++)
        total_pairs += count(X[i], Y, n, NoOfY);

    return total_pairs;
}
```

~ Himanshi Sehgal

Inversion of an array

For an array, inversion count indicates how far (or close) the array is from being sorted. If array is already sorted then the inversion count is 0. If an array is sorted in the reverse order then the inversion count is the maximum.
Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.

```
Input: N = 5, arr[] = {2, 4, 1, 3,
 5}
Output: 3
Explanation: The sequence 2, 4, 1,
 3, 5
has three inversions (2, 1), (4, 1
), (4, 3).
```

Example 2:

```
Input: N = 5
arr[] = {2, 3, 4, 5, 6}
Output: 0
Explanation: As the sequence is al
ready
sorted so there is no inversion co
unt.
```

Example 3:

```
Input: N = 3, arr[] = {10, 10, 10}
Output: 0
Explanation: As all the elements o
f array
are same, so there is no inversion
```

First approach

Use nested for loops and find the pairs that satisfies the condition

$a[i] > a[j]$ & $i < j$

eg arr = [2, 4, 1]

if i = 1 & j = 2

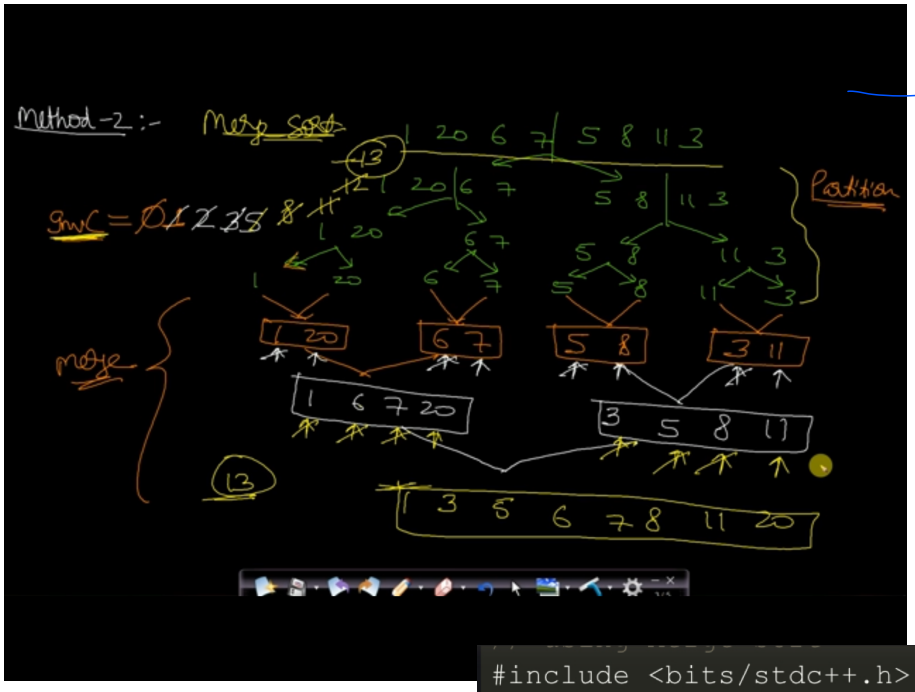
arr[i] = 4 , arr[j] = 1

one pair → (4, 1)

2nd pair → (2, 1)

Output = 2 → (count return)

~ Himanshi Sehgal



→ 2nd Method
By using
Merge Sort
for better TC

```cpp
#include <bits/stdc++.h>
using namespace std;

int _mergeSort(int arr[], int temp[], int left, int right);
int merge(int arr[], int temp[], int left, int mid,
          int right);

/* This function sorts the
   input array and returns the
number of inversions in the array */
int mergeSort(int arr[], int array_size)
{
    int temp[array_size];
    return _mergeSort(arr, temp, 0, array_size - 1);
}

/* 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;
}
```

declare functions

make all jo partitions jo hr green color se hme merge h

jo bé partitions bne h unhe merge kro

~ Himanshi Sehgal

```cpp
int merge(int arr[], int temp[], int left, int mid,
          int right)
{
    int i, j, k;
    int inv_count = 0;

    i = left; /* i is index for left subarray*/
    j = mid; /* j is index for right subarray*/
    k = left; /* k is index for resultant merged subarray*/
    while ((i <= mid - 1) && (j <= right)) {
        if (arr[i] <= arr[j]) {
            temp[k++] = arr[i++];
        }
        else {
            temp[k++] = arr[j++];

            /* this is tricky -- see above
            explanation/diagram for merge()*/
            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;
}

// Driver code
int main()
{
    int arr[] = { 1, 20, 6, 4, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int ans = mergeSort(arr, n);
    cout << " Number of inversions are " << ans;
    return 0;
}
```

```
class Solution {
public:
    int merge(vector<int>& nums, int low, int mid, int high){
        int count=0;
        int j=mid+1;

        for(int i=low; i<=mid; i++){
            while(j<=high and nums[i] > 2LL * nums[j]){
                j++;
            }
            count += j-(mid+1);
        }

        vector<int> temp;
        int left =low, right= mid+1;

        while(left<=mid and right<=high){
            if(nums[left] > nums[right]){
                temp.push_back(nums[right]);
                right++;
            }else{
                temp.push_back(nums[left]);
                left++;
            }
        }

        while(left<=mid){
            temp.push_back(nums[left]);
            left++;
            }

        while(right<=high){
            temp.push_back(nums[right]);
            right++;
            }

        for(int i=low; i<=high; i++){
            nums[i] = temp[i-low];
        }

        return count;

    }
```

```
    int mergesort(vector<int>& nums, int low , int high){
        if(low>=high) return 0;
        int mid = low+(high-low)/2;
        int inv = mergesort(nums,low, mid);
        inv += mergesort(nums,mid+1,high);
        inv += merge(nums,low,mid,high);
        return inv;
    }
    int reversePairs(vector<int>& nums) {
        return mergesort(nums,0, nums.size()-1);
    }
};
```

~ Himanshi Sehgal

Equilibrium point in an array

Example :

Input: *A[] = {-7, 1, 5, 2, -4, 3, 0}*

Output: *3*

3 is an equilibrium index, because:

A[0] + A[1] + A[2] = A[4] + A[5] + A[6]

Input: *A[] = {1, 2, 3}*

Output: *-1*

Given an array A of n positive numbers. The task is to find the first Equilibium Point in the array.
Equilibrium Point in an array is a position such that the sum of elements before it is equal to the sum of elements after it.

Approach

Method 2 (Tricky and Efficient)

The idea is to get the total sum of the array first. Then Iterate through the array and keep updating the left sum which is initialized as zero. In the loop, we can get the right sum by subtracting the elements one by one. Thanks to Sambasiva for suggesting this solution and providing code for this.

```
1) Initialize leftsum  as 0
2) Get the total sum of the array as sum
3) Iterate through the array and for each index i, do f
    a)  Update sum to get the right sum.
            sum = sum - arr[i]
        // sum is now right sum
    b) If leftsum is equal to sum, then return current
        // update leftsum for next iteration.
    c) leftsum = leftsum + arr[i]
4) return -1
// If we come out of loop without returning then
// there is no equilibrium index
```

The image below shows the dry run of the above approach:


```cpp
// C++ program to find equilibrium
// index of an array
#include <bits/stdc++.h>
using namespace std;

int equilibrium(int arr[], int n)
{
    int sum = 0; // initialize sum of whol
    int leftsum = 0; // initialize leftsum

    /* Find sum of the whole array */
    for (int i = 0; i < n; ++i)
        sum += arr[i];

    for (int i = 0; i < n; ++i)
    {
        sum -= arr[i]; // sum is now right

        if (leftsum == sum)
            return i;

        leftsum += arr[i];
    }

    /* If no equilibrium index found, then
    return -1;
}
```

~ Himanshi Sehgal

Minimum platforms

Given arrival and departure times of all trains that reach a railway station. Find the minimum number of platforms required for the railway station so that no train is kept waiting. Consider that all the trains arrive on the same day and leave on the same day. Arrival and departure time can never be the same for a train but we can have arrival time of one train equal to departure time of the other. At any given instance of time, same platform can not be used for both departure of a train and arrival of another train. In such cases, we need different platforms.

Input: n = 6
arr[] = {0900, 0940, 0950, 1100, 1500, 1800}
dep[] = {0910, 1200, 1120, 1130, 1900, 2000}
Output: 3
Explanation:
Minimum 3 platforms are required to
safely arrive and depart all trains.

Example 2:

Input: n = 3
arr[] = {0900, 1100, 1235}
dep[] = {1000, 1200, 1240}
Output: 1
Explanation: Only 1 platform is required to
safely manage the arrival and departure
of all trains.

Solution in O(n*log(n)) time, classical problem for intervals.

```
int minParkingSpace(vector<vector<int>> parkingTime){

    map<int,int>mymap;

    for(int i=0;i<parkingTime.size();i++){
        mymap[parkingTime[i][0]]++;
        mymap[parkingTime[i][1]]--;
    }

    int ans=0;
    int count=0;
    for(auto it=mymap.begin();it!=mymap.end();++it){
        count+=it->second;
        ans=max(ans,count);
    }
    return ans;
}

int main(void){

    cout<<minParkingSpace({{5,10}, {0,20}, {25,40}, {35,45}})<<endl;

    return 0;
}
```

1. Take a hashmap
2. Increment the value by 1 if its arrival time
3. decrement the value by 1 if its departure time
4. Iterate in map, maintain max count because max count hi btayega that ek particular time pei kitni trains platform pei thi

~ Himanshi Sehgal

Pythagorean Triplet

Basic Accuracy: 49.17% Submissions: 19946 Points: 1

Given an array **arr** of **N** integers, write a function that returns true if there is a triplet (a, b, c) that satisfies $a^2 + b^2 = c^2$, otherwise false.

Example 1:

```
Input:
N = 5
Arr[] = {3, 2, 4, 6, 5}
Output: Yes
Explanation: a=3, b=4, and c=5 forms a
pythagorean triplet.
```

Example 2:

```
Input:
N = 3
Arr[] = {3, 8, 5}
Output: No
```

Method -4:Using STL

Approach:

The problem can be solved using ordered maps and unordered maps. There is no need to store the elements in an ordered manner so implementation by an unordered map is faster. We can use the unordered map to mark all the values of the given array. Using two loops, we can iterate for all the possible combinations of a and b, and then check if there exists the third value c. If there exists any such value, then there is a Pythagorean triplet.

Below is the implementation of the above approach:

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

// Returns true if there is Pythagorean triplet in
// ar[0..n-1]
bool checkTriplet(int arr[], int n)
{
    // initializing unordered map with key and value as
    // integers
    unordered_map<int, int> umap;

    // Increase the count of array elements in unordere
    for (int i = 0; i < n; i++)
        umap[arr[i]] = umap[arr[i]] + 1;

    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            // calculating the squares of two elements
            // integer and float
            int p = sqrt(arr[i] * arr[i] + arr[j] * arr
            float q
                = sqrt(arr[i] * arr[i] + arr[j] * arr[j

            // Condition is true if the value is same i
            // integer and float and also the value is
            // present in unordered map
            if (p == q && umap[p] != 0)
                return true;
        }
    }

    // If we reach here, no triplet found
    return false;
}

// Driver Code
```

~ Himanshi Sehgal

Chocolate distribution problem

Approach

1. Apply sort
2. for each sub subarrays to of size m & minm difference nikalo

Given an array of n integers where each value represents the number of chocolates in a packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that:

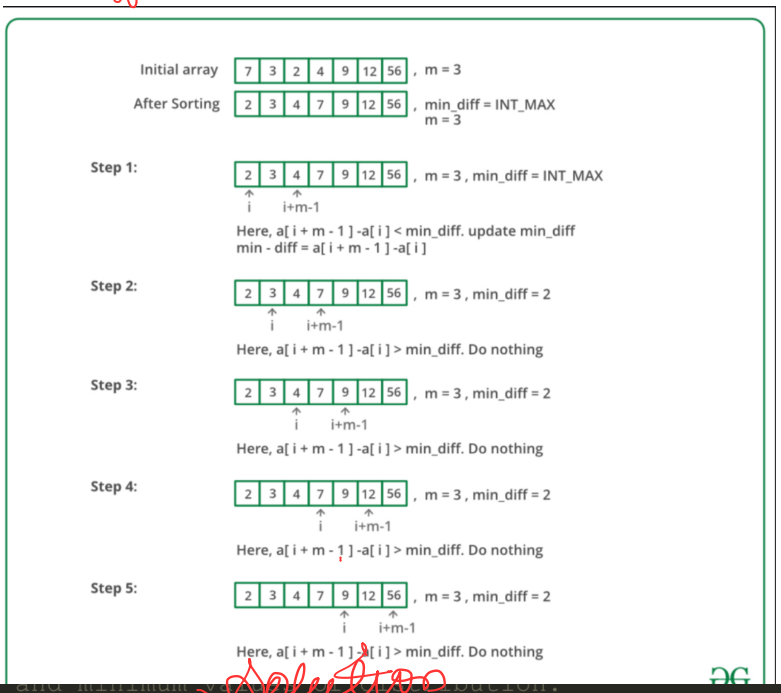
1. Each student gets one packet.
2. The difference between the number of chocolates in the packet with maximum chocolates and packet with minimum chocolates given to the students is minimum.

Examples:

Input : *arr[] = {7, 3, 2, 4, 9, 12, 56} , m = 3*
Output: *Minimum Difference is 2*
Explanation:
We have seven packets of chocolates and we need to pick three packets for 3 students If we pick 2, 3 and 4, we get the minimum difference between maximum and minimum packet sizes.

Input : *arr[] = {3, 4, 1, 9, 56, 7, 9, 12} , m = 5*
Output: *Minimum Difference is 6*
Explanation:
The set goes like 3,4,7,9,9 and the output is 9-3 = 6

Input : *arr[] = {12, 4, 7, 9, 2, 23, 25, 41, 30, 40, 28, 42, 30, 44, 48, 43, 50} , m = 7*
Output: *Minimum Difference is 10*
Explanation:
We need to pick 7 packets. We pick 40, 41, 42, 44, 48, 43 and 50 to minimize difference between maximum and minimum.



Solution

```
int findMinDiff(int arr[], int n, int m)
{
    // if there are no chocolates or number
    // of students is 0
    if (m == 0 || n == 0)
        return 0;

    // Sort the given packets
    sort(arr, arr + n);

    // Number of students cannot be more than
    // number of packets
    if (n < m)
        return -1;

    // Largest number of chocolates
    int min_diff = INT_MAX;

    // Find the subarray of size m such that
    // difference between last (maximum in case
    // of sorted) and first (minimum in case of
    // sorted) elements of subarray is minimum.

    for (int i = 0; i + m - 1 < n; i++) {
        int diff = arr[i + m - 1] - arr[i];
        if (diff < min_diff)
            min_diff = diff;
    }
    return min_diff;
}
```

~ Himanshi Sehgal

Convert array into Zig-Zag fashion

Basic Accuracy: 51.25% Submissions: 13626
Points: 1

Given an array **Arr** (distinct elements) of size **N**. Rearrange the elements of array in zig-zag fashion. The converted array should be in form **a < b > c < d > e < f.** The relative order of elements is same in the output **i.e** you have to iterate on the original array only.

4 3 7 8 6 2 1
3 4 7 8 6 2 1
3 7 4 8 6 2 1
3 7 4 8 2 6 1

Example 1:

```
Input:
N = 7
Arr[] = {4, 3, 7, 8, 6, 2, 1}
Output: 3 7 4 8 2 6 1
Explanation: 3 < 7 > 4 < 8 > 2 < 6
 > 1
```

1. Sort Array
2. Traverse the array from index 1 to n-1 &
   increase value of index by 2
3. While traversing the array, Swap (i) &
   (i+1) position data in array

~ Himanshi Sehgal

Spirally traversing a matrix

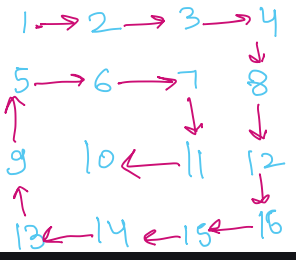
```
Input:  1   2   3   4
        5   6   7   8
        9   10  11  12
        13  14  15  16
Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
Explanation: The output is matrix in spiral format

Input:  1   2   3   4  5   6
        7   8   9   10  11  12
        13  14  15 16  17  18
Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10
Explanation :The output is matrix in spiral format
```

Matrix:

```
1→ 2 → 3 → 4
                ↓
5 → 6 → 7   8   ↓
↑       ↓       ↓
9   10 ← 11  12  ↓
↑               ↓
13← 14 ← 15 ← 16
```

Output:

1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10



mr = 4 cl = 4
row = 0 col = 0
phle col = 0 kei sb print krado
row -

- **Algorithm:**
  1. Create and initialize variables k – starting row index, m – ending row index, l – starting column index, n – ending column index
  2. Run a loop until all the squares of loops are printed.
  3. In each outer loop traversal print the elements of a square in a clockwise manner.
  4. Print the top row, i.e. Print the elements of the kth row from column index l to n, and increase the count of k.
  5. Print the right column, i.e. Print the last column or n-1th column from row index k to m and decrease the count of n.
  6. Print the bottom row, i.e. if k < m, then print the elements of m-1th row from column n-1 to l and decrease the count of m
  7. Print the left column, i.e. if l < n, then print the elements of lth column from m-1th row to k and increase the count of l.

```cpp
#include <bits/stdc++.h>
using namespace std;
#define R 3
#define C 6

void spiralPrint(int m, int n, int a[R][C])
{
    int i, k = 0, l = 0;

    /* k - starting row index
        m - ending row index
        l - starting column index
        n - ending column index
        i - iterator
    */

    while (k < m && l < n) {
        /* Print the first row from
                the remaining rows */
        for (i = l; i < n; ++i) {
            cout << a[k][i] << " ";
        }
        k++;

        /* Print the last column
         from the remaining columns */
        for (i = k; i < m; ++i) {
            cout << a[i][n - 1] << " ";
        }
        n--;

        /* Print the last row from
                the remaining rows */
        if (k < m) {
            for (i = n - 1; i >= l; --i) {
                cout << a[m - 1][i] << " ";
            }
            m--;
        }

        /* Print the first column from
                    the remaining columns */
        if (l < n) {
            for (i = m - 1; i >= k; --i) {
                cout << a[i][l] << " ";
            }
            l++;
        }
    }
}
```

~ Himanshi Sehgal

Largest number formed from an array

Largest Number formed from an Array

Medium Accuracy: 50.0% Submissions: 22001 Points: 4

Given a list of non negative integers, arrange them in such a manner that they form the largest number possible.The result is going to be very large, hence return the result in the form of a string.

Example 1:

```
Input:
N = 5
Arr[] = {3, 30, 34, 5, 9}
Output: 9534330
Explanation: Given numbers are {3, 30, 34,
5, 9}, the arrangement 9534330 gives the
largest value.
```

Example 2:

```
Input:
N = 4
Arr[] = {54, 546, 548, 60}
Output: 6054854654
Explanation: Given numbers are {54, 546,
548, 60}, the arrangement 6054854654
gives the largest value.
```

make a function and use me ke element ko string consider kro phir and y x, apply concat & fir unhe compare kro and fir greater wala return kro

y sb ek comparator function mei hai to sort ke sath lgado ise taki simultaneously we are sorting

we

so the ans. will be in O(n log n)

```cpp
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// A comparison function which
// is used by sort() in
// printLargest()
int myCompare(string X, string Y)
{
    // first append Y at the end of X
    string XY = X.append(Y);

    // then append X at the end of Y
    string YX = Y.append(X);

    // Now see which of the two
    // formed numbers is greater
    return XY.compare(YX) > 0 ? 1 : 0;
}

// The main function that prints
// the arrangement with the
// largest value. The function
// accepts a vector of strings
void printLargest(vector<string> arr)
{

    // Sort the numbers using
    // library sort function. The
    // function uses our comparison
    // function myCompare() to
    // compare two strings. See
    // http://www.cplusplus.com/reference/
    // algorithm/sort/
    // for details
    sort(arr.begin(), arr.end(), myCompare);

    for (int i = 0; i < arr.size(); i++)
        cout << arr[i];
}

// Driver code
int main()
{
    vector<string> arr;

    // output should be 6054854654
    arr.push_back("54");
    arr.push_back("546");
    arr.push_back("548");
    arr.push_back("60");
    printLargest(arr);

    return 0;
}
```

imp

~ Himanshi Sehgal