Time Complexity

Complexity

Send Feedback

Which complexity of an algorithm quantifies the amount of memory taken by an algorithm.

Options

This problem has only one correct answer

- ○ Time complexity
- ◉ Space complexity

Hurray! Correct Answer

Number of operations

Send Feedback

What are the number of operations for the following function?

```
public static void func(int n)
{
   int sum=0;
   for(int i=2;i<n;i+=2)
      sum+=i;
   System.out.println(sum);
}
```

Options

This problem has only one correct answer

- ◉ k1+k2(n)
- ○ k1+k2
- ○ k1+k2(n^2)
- ○ k1+k2(logn)

Hurray! Correct Answer

Number of operations

Send Feedback

What are the number of operations for the following function?

```
public static void func(int n)
{
   int sum=0;
   for(int i=1;i<n*n;i++)
      sum+=i;
   System.out.println(sum);
}
```

Options

This problem has only one correct answer

- ○ k1+k2(n)
- ○ k1+k2(logn)
- ◉ k1+k2(n^2)
- ○ k1(n)+k2(logn)

Hurray! Correct Answer

Time complexity of a code

Send Feedback

What will be the Time Complexity of following code in terms of 'n' ?

```
public static void func(int n)
{
   int sum=0;
   for(int i=1;i<n;i++)
      {
         for(;i<n*n;i++)
         {
            sum+=i;
         }
      }
   System.out.println(sum);
}
```

Options

This problem has only one correct answer

- O(n)
- O(n^2)
- O(n^3)
- O(n^4)

Hurray! Correct Answer

Time complexity of a code

Send Feedback

What will be the Time Complexity of following code in terms of 'n' ?

```
public static void func(int n)
{
   int sum=0;
   for(int i=1;i<n;i++)
      {
         for(int j=1;j<n*n;j++)
         {
            sum+=i;
         }
      }
   System.out.println(sum);
}
```

Options

This problem has only one correct answer

- O(n)
- O(n^2)
- O(n^3)
- O(n^4)

Hurray! Correct Answer

Time complexity

Send Feedback

Choose the correct option.

A) Linear Search	1. O(logn)
B) Bubble sort	2. O(n)
C) Binary Search	3. O(n^2)

Options

This problem has only one correct answer

- A->1 B->3 C->2
- A->3 B->2 C->1
- A->3 B->1 C->2
- (selected) A->2 B->3 C->1

Hurray! Correct Answer

Time complexity of a code

Send Feedback

What will be the Time Complexity of following code in terms of 'n' ?

```
public static void func(int n)
{
   int sum=0;
   for(int i=1;i<n;i++)
      {
         for(int j=1;j<=i;j++)
         {
            sum+=i;
         }
      }
   System.out.println(sum);
}
```

Options

This problem has only one correct answer

- O(n)
- (selected) O(n^2)
- O(nlogn)
- O(n^3)

Hurray! Correct Answer

Time complexity of a code

Send Feedback

What will be the Time Complexity of following code in terms of 'n' ?

```
public static void func(int n)
{
   int sum=0;
   for(int i=1;i<n;i*=2)
      {
            sum+=i;
      }
   System.out.println(sum);
}
```

Options

This problem has only one correct answer

- (•) O(log n(base 2))
- ( ) O(n)
- ( ) O(ln(n))
- ( ) O(n^2)

Hurray! Correct Answer

Merging two Sorted arrays

Send Feedback

What is the time complexity for merging two sorted arrays?

- size of arrays are n and m.

Options

This problem has only one correct answer

- ( ) O(n*m)
- (•) O(n+m)
- ( ) O(n)
- ( ) O(m)

Hurray! Correct Answer

Array Intersection

Send Feedback

You have been given two integer arrays/list(ARR1 and ARR2) of size N and M, respectively. You need to print their intersection; An intersection for this problem can be defined when both the arrays/lists contain a particular value or to put it in other words, when there is a common value that exists in both the arrays/lists.

Note :

Input arrays/lists can contain duplicate elements.

The intersection elements printed would be in ascending order.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains an integer 'N' representing the size of the first array/list.

The second line contains 'N' single space separated integers representing the elements of the first the array/list.

The third line contains an integer 'M' representing the size of the second array/list.

The fourth line contains 'M' single space separated integers representing the elements of the second array/list.

Output format :

For each test case, print the intersection elements in a row, separated by a single space.

Output for every test case will be printed in a separate line.

Constraints :

1 <= t <= 10^2
0 <= N <= 10^6
0 <= M <= 10^6

Time Limit: 1 sec

Sample Input 1 :

```
2
6
2 6 8 5 4 3
4
2 3 4 7
2
10 10
1
10
```

Sample Output 1 :

```
2 3 4
10
```

Sample Input 2 :

```
1
4
2 6 1 2
5
1 2 3 4 2
```

Sample Output 2 :

```
1 2 2

Explanation for Sample Output 2 :

Since, both input arrays have two '2's, the intersection of the arrays
also have two '2's. The first '2' of first array matches with the first
'2' of the second array. Similarly, the second '2' of the first array
matches with the second '2' if the second array.


import java.util.Arrays;

public class Solution {
   public static void mergeSort(int[] input) {
       // Write your code here
       if (input.length == 1 || input.length == 0)
           return;

       int mid = input.length % 2 == 0 ? input.length / 2 : input.length /
2 + 1;

       int lefthalf[] = new int[mid];
       int righthalf[] = new int[input.length - mid];

       for (int i = 0; i < mid; i++) {
           lefthalf[i] = input[i];
       }
       for (int temp = 0, i = mid; i < input.length; temp++, i++) {
           righthalf[temp] = input[i];
       }

       mergeSort(lefthalf);
       mergeSort(righthalf);
       // int res[] = new int[lefthalf.length*2];
       int i = 0, j = 0, k = 0;

       while (i < lefthalf.length && j < righthalf.length) {
           if (lefthalf[i] < righthalf[j])
               input[k++] = lefthalf[i++];
           else
               input[k++] = righthalf[j++];

       }
       while (i < lefthalf.length)
           input[k++] = lefthalf[i++];

       while (j < righthalf.length)
           input[k++] = righthalf[j++];

   }

   public static int partition(int[] input, int si, int ei) {
```

```
        int pivotelement = input[si];
        int count = 0;
        for (int i = si; i <= ei; i++) {
            if (input[i] < pivotelement)
                count++;
        }

        int temp = input[si + count];
        input[si + count] = pivotelement;
        input[si] = temp;

        int i = si, j = ei;
        while (i < j) {
            if (input[i] < pivotelement)
                i++;
            else if (input[j] >= pivotelement)
                j--;
            else {
                temp = input[j];
                input[j] = input[i];
                input[i] = temp;
                i++;
                j--;
            }
        }
        return si + count;
    }

    public static void sort(int[] input, int si, int ei) {
        if (si >= ei) {
            return;
        }

        int pivotelement = partition(input, si, ei);
        sort(input, si, pivotelement - 1);
        sort(input, pivotelement + 1, ei);
    }

    public static void intersection(int[] arr1, int[] arr2) {

        if (arr1.length < arr2.length) {
            mergeSort(arr1);
            for (int i = 0; i < arr1.length; i++) {
                for (int j = 0; j < arr2.length; j++) {
                    if (arr1[i] == arr2[j]) {
                        System.out.print(arr1[i] + " ");
                        arr2[j] = Integer.MIN_VALUE;
                        break;
                    }
                }
            }
        } else {
            sort(arr2, 0, arr2.length - 1);
            for (int j = 0; j < arr2.length; j++) {
```

```
                for (int i = 0; i < arr1.length; i++) {
                    if (arr1[i] == arr2[j]) {
                        System.out.print(arr1[i] + " ");
                        arr1[i] = Integer.MIN_VALUE;
                        // System.out.println(Arrays.toString(arr1));
                        break;
                    }
                }
            }
        }
    }
}
```

Array Equilibrium Index
Send Feedback
For a given array/list(ARR) of size 'N,' find and return the 'Equilibrium Index' of the array/list.
Equilibrium Index of an array/list is an index 'i' such that the sum of elements at indices [0 to (i - 1)] is equal to the sum of elements at indices [(i + 1) to (N-1)]. One thing to note here is, the item at the index 'i' is not included in either part.
If more than one equilibrium indices are present, then the index appearing first in left to right fashion should be returned. Negative one(-1) if no such index is present.
Example:

Let's consider an array/list Arr = [2, 3, 10, -10, 4, 2, 9] of size, N = 7.

There exist three equilibrium indices, one at 2, another at 3, and another at 5.

At index 2, the sum of all the elements to the left, [2 + 3] is 5, and the elements to its right, [-10 + 4 + 2 + 9] is also 5. Hence index 2 is an equilibrium index according to the condition we want to achieve. Mind it that we haven't included the item at index 2, which is 10, to either of the parts.

Similarly, we can see at index 3 and 5, the elements to its left sum up to 15 and 9 respectively and to the right, sum up to 15 and 9 respectively either.

Hence the answer would be 3.

Input Format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of each test case or query contains an integer 'N' representing the size of the first array/list.

The second line contains 'N' single space separated integers representing the elements of the array/list

Output Format :

For each test case, print the 'Equilibrium Index'.

Output for every test case will be printed in a separate line.

Constraints :

1 <= t <= 10^2
0 <= N <= 10^6

Time Limit: 1 sec

Sample Input 1 :

```
1
5
1 4 9 3 2
```

Sample Output 1 :

```
2
```

Sample Input 2 :

```
2
3
1 4 6
3
1 -1 4
```

Sample Output 2 :

```
-1
2
```

```java
public class Solution {

   public static int arrayEquilibriumIndex(int[] arr){

        int pivot = arr.length/2;
        int sum=0, res=0, lsum = 0, rsum=0;

        if(arr.length==0)
            return -1;

        for(int i=0; i<arr.length; i++){
            sum+=arr[i];
        }

        rsum = sum - arr[0];
        lsum = 0;
```

```
        for(int i=1; i<arr.length; i++){
            rsum-=arr[i];
            lsum+=arr[i-1];

            if(rsum == lsum){
                res = i;
                break;
            }
        }
        if(res!=0)
            return res;
        else
            return -1;
    }
}
```

Find the *Unique Element*
Send Feedback
You have been given an integer array/list(ARR) of size N. Where *N* is equal to [2M + 1].
Now, in the given array/list, 'M' numbers are present twice and one number is present only once.
You need to find and return that number which is unique in the array/list.
Note:

Unique element is always present in the array/list according to the given condition.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

First line of each test case or query contains an integer 'N' representing the size of the array/list.

Second line contains 'N' single space separated integers representing the elements in the array/list.

Output Format :

For each test case, print the unique element present in the array.

Output for every test case will be printed in a separate line.

Constraints :

1 <= t <= 10^2
0 <= *N* <= 10^6

Time Limit: 1 sec

Sample Input 1:

```
1
7
2 3 1 6 3 6 2

Sample Output 1:

1

Sample Input 2:

2
5
2 4 7 2 7
9
1 3 1 3 6 6 7 10 7

Sample Output 2:

4
10

public class Solution {

  public static int findUnique(int[] arr) {

        int ans=0;
        for(int i=0; i<arr.length; i++){
            ans^=arr[i];
        }
        return ans;
    }
}
```

```
Duplicate in array
Send Feedback
You have been given an integer array/list(ARR) of size N which contains
numbers from 0 to (N - 2). Each number is present at least once. That is,
if N = 5, the array/list constitutes values ranging from 0 to 3, and among
these, there is a single integer value that is present twice. You need to
find and return that duplicate number present in the array.
Note :

Duplicate number is always present in the given array/list.

Input format :

The first line contains an Integer 't' which denotes the number of test
cases or queries to be run. Then the test cases follow.

First line of each test case or query contains an integer 'N' representing
the size of the array/list.

Second line contains 'N' single space separated integers representing the
elements in the array/list.
```

Output Format :

For each test case, print the duplicate element in the array/list.

Output for every test case will be printed in a separate line.

Constraints :

1 <= t <= 10^2
0 <= *N* <= 10^6

Time Limit: 1 sec

Sample Input 1:

```
1
9
0 7 2 5 4 7 1 3 6
```

Sample Output 1:

```
7
```

Sample Input 2:

```
2
5
0 2 1 3 1
7
0 3 1 5 4 3 2
```

Sample Output 2:

```
1
3
```

```
public class Solution {

   public static int findDuplicate(int[] arr) {
       //Your code goes here

       int n = arr.length-2;
       int nsum = (n*(n+1))/2;
       int sum=0;
       for(int i=0; i<n+2; i++){
           sum+=arr[i];
       }
       return sum-nsum;

   }
}
```

Pair sum in array

Send Feedback
You have been given an integer array/list(ARR) and a number 'num'. Find and return the total number of pairs in the array/list which sum to 'num'.
Note:

Given array/list can contain duplicate elements.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

First line of each test case or query contains an integer 'N' representing the size of the first array/list.

Second line contains 'N' single space separated integers representing the elements in the array/list.

Third line contains an integer 'num'.

Output format :

For each test case, print the total number of pairs present in the array/list.

Output for every test case will be printed in a separate line.

Constraints :

1 <= t <= 10^2
0 <= N <= 10^4
0 <= num <= 10^9

Time Limit: 1 sec

Sample Input 1:

```
1
9
1 3 6 2 5 4 3 2 4
7
```

Sample Output 1:

```
7
```

Sample Input 2:

```
2
9
1 3 6 2 5 4 3 2 4
12
6
2 8 10 5 -2 5
```

```
10
```

Sample Output 2:

```
0
2
```

Explanation for Input 2:

Since there doesn't exist any pair with sum equal to 12 for the first query, we print 0.

For the second query, we have 2 pairs in total that sum up to 10. *They* are, (2, 8) and (5, 5).

```
import java.util.Arrays;

public class Solution {

   public static void mergesort(int a[]){

       if(a.length==1)
           return;

       int mid = a.length/2;
       int l[] = new int[mid];
       int r[] = new int[a.length-mid];

       for(int i=0; i<mid; i++){
           l[i] = a[i];
       }

       for(int index = 0, i=mid; i<a.length; index++, i++){
           r[index] = a[i];
       }

       mergesort(l);
       mergesort(r);

       int i=0, j=0, k=0;
       while(i<l.length && j<r.length){
           if(l[i]<r[j])
               a[k++] = l[i++];
           else
               a[k++] = r[j++];
       }
       while(i<l.length)
               a[k++] = l[i++];
       while(j<r.length)
           a[k++] = r[j++];


   }
```

```java
   public static int pairSum(int[] arr, int num) {
       //Your code goes here
       if(arr.length==0)
           return 0;
       mergesort(arr);

       boolean flag = true;
       for(int i=1; i<arr.length; i++){
           if(arr[i]!=arr[i-1])
               flag = false;
       }
       if(flag){
           int n=arr.length;
           return (n*(n-1))/2;
       }

       int i=0, j=arr.length-1, count = 0;
       while(i<j){
           if(arr[i]+arr[j]==num){

               int counti=0, countj=0;
               i++;
               j--;
               int tempi=i;
               int tempj=j;
               while(arr[i-1]==arr[tempi] && tempi<=j){
                   counti++;
                   // System.out.println("counti = "+counti+" i =
"+tempi+" j = "+j);
                   tempi++;
               }
               while(arr[j+1]==arr[tempj] && tempj>=i){
                   countj++;
                   // System.out.println("countj = "+countj+" j =
"+tempj+" i = "+i);
                   tempj--;
               }
               if(arr[i-1]!=arr[j+1])
                   count += ++counti * ++countj;
               else
                   count += counti * countj;
                   // System.out.println("count "+count +"counti+1 "+
++counti+" countj+1+ "+countj+1);

               i = tempi;
               j = tempj;

           }
           else if(arr[i]+arr[j]>num)
               j--;
           else
               i++;
       }
```

```
        return count;
    }
}
```

Triplet sum
Send Feedback
You have been given a random integer array/list(ARR) and a number X. Find and return the triplet(s) in the array/list which sum to X.
Note :

Given array/list can contain duplicate elements.

Input format :

The first line contains an Integer 't' which denotes the number of test cases or queries to be run. Then the test cases follow.

First line of each test case or query contains an integer 'N' representing the size of the first array/list.

Second line contains 'N' single space separated integers representing the elements in the array/list.

Third line contains an integer 'X'.

Output format :

For each test case, print the total number of triplets present in the array/list.

Output for every test case will be printed in a separate line.

Constraints :

1 <= t <= 10^2
0 <= *N* <= 10^3
0 <= *X* <= 10^9

Time Limit: 1 sec

Sample Input 1:

```
1
7
1 2 3 4 5 6 7
12
```

Sample Output 1:

```
5
```

Sample Input 2:

```
2
```

```
7
1 2 3 4 5 6 7
19
9
2 -5 8 -6 0 5 10 11 -3
10

Sample Output 2:

0
5


Explanation for Input 2:

Since there doesn't' exist any triplet with sum equal to 19 for the first query, we print 0.

For the second query, we have 5 triplets in total that sum up to 10. They are, (2, 8, 0), (2, 11, -3), (-5, 5, 10), (8, 5, -3) and (-6, 5, 11)

import java.util.*;

public class Solution {

   public static int tripletSum(int arr[], int num) {
       Arrays.sort(arr);
       int n = arr.length;
       int count = 0;
       for (int i = 0; i < n; i++) {
           int temp = num - arr[i];
           // finding pairs of temp;
           count += findPairHelper(arr, i + 1, n - 1, temp);
       }

       return count;
   }

   private static int findPairHelper(int arr[], int sI, int eI, int key) {
       int numPair = 0;
       while (sI < eI) {
           if (arr[sI] + arr[eI] > key)/*
                                        * Since array is sorted so while
adding the element at ith index and jth index
                                        * and if you get sum greater than
k then you need to decrement j because j is
                                        * at end index
                                        */
               eI--;
           else if (arr[sI] + arr[eI] < key)/*
                                             * while adding the element at
ith index at the jth index if you get sum
                                             * less than key then you need
to increment the i because i is at starting
```

```
                                                * index
                                                */
                sI++;
            else if (arr[sI] == arr[eI]) {
                // if both elements are same;
                /*
                 * if the elment at the ith index is equal to element at
the jth index then all
                 * the elements
                 * in the middle are also equal for example : 4, 4, 4, 4,
now element at
                 * startIndex is 4 and element at endIndex is
                 * if you see carefully all the elements in middle also
equal and this is
                 * because our array is sorted in ascending
                 * order so whenever we get such input we need total
combination of pair that
                 * are possible out of n equal elements
                 * that is Nc2 so we use here Ncr formula to find totoal
combination of pair out
                 * of n equal elements
                 */
                int num = (eI - sI) + 1;
                numPair += ncr(num, 2);
                return numPair;
            } else {

                int startElement = arr[sI];
                int endElement = arr[eI];
                int tempStartIndex = sI + 1;
                int tempEndIndex = eI - 1;

                while (tempStartIndex <= tempEndIndex &&
arr[tempStartIndex] == startElement)
                    tempStartIndex++;

                while (tempStartIndex <= tempEndIndex && arr[tempEndIndex]
== endElement)
                    tempEndIndex--;

                int totalElementFromStart = tempStartIndex - sI;
                int totalElementFromEnd = eI - tempEndIndex;

                numPair += (totalElementFromEnd * totalElementFromStart);

                sI = tempStartIndex;
                eI = tempEndIndex;
            }
        }

        return numPair;
    }

    public static int ncr(int num, int r) {
```

```
        int ans = (factorial(num) / factorial(num - r)) / factorial(r);
        return ans;

    }

    public static int factorial(int number) {
        int ans = 1;
        for (int i = 1; i <= number; i++)
            ans *= i;
        return ans;
    }
}
```

```
Rotate array
Send Feedback
You have been given a random integer array/list(ARR) of size N. Write a
function that rotates the given array/list by D elements(towards the
left).
Note:

Change in the input array/list itself. You don't need to return or print
the elements.

Input format :

The first line contains an Integer 't' which denotes the number of test
cases or queries to be run. Then the test cases follow.

First line of each test case or query contains an integer 'N' representing
the size of the array/list.

Second line contains 'N' single space separated integers representing the
elements in the array/list.

Third line contains the value of 'D' by which the array/list needs to be
rotated.

Output Format :

For each test case, print the rotated array/list in a row separated by a
single space.

Output for every test case will be printed in a separate line.

Constraints :

1 <= t <= 10^4
0 <= N <= 10^6
0 <= D <= N
Time Limit: 1 sec

Sample Input 1:

1
```

```
7
1 2 3 4 5 6 7
2

Sample Output 1:

3 4 5 6 7 1 2

Sample Input 2:

2
7
1 2 3 4 5 6 7
0
4
1 2 3 4
2

Sample Output 2:

1 2 3 4 5 6 7
3 4 1 2

public class Solution {

   public static void rotate(int[] arr, int d) {
       //Your code goes here
       int temp[] = new int[d];
       int i=0;

       while(i<d){
           temp[i] = arr[i++];
       }
       int j=0;
       while(i<arr.length){
           arr[j++] = arr[i++];
       }
       i = 0;
       while(i<d){
           arr[j++] = temp[i++];
       }
   }

}
```