

*//User function Template for Java*

class Compute {

    public static void reverse(int[] arr, int i, int j){

        while(i<=j){

            int temp = arr[i];

            arr[i] = arr[j];

            arr[j] = temp;

            i++;

            j--;

        }

    }

    public void rotate(int arr[], int n)

    {

*// first method :*

*// 1,2, 3, 4, 5  =>reverse till second last => 4, 3, 2, 1, 5*

*//=> reverse all => 5,1,2,3,4*

*// op: 5,1,2,3,4*

*// reverse(arr,0,arr.length-2);*

*// reverse(arr,0,arr.length-1);*

*//second method*

*/\**

*Following are steps.*

*1) Store last element in a variable say x.*

*2) Shift all elements one position ahead.*

*3) Replace first element of array with x.*

*\*/*

        int temp = arr[arr.length-1];

        for(int i=arr.length-1;i>0;i--){

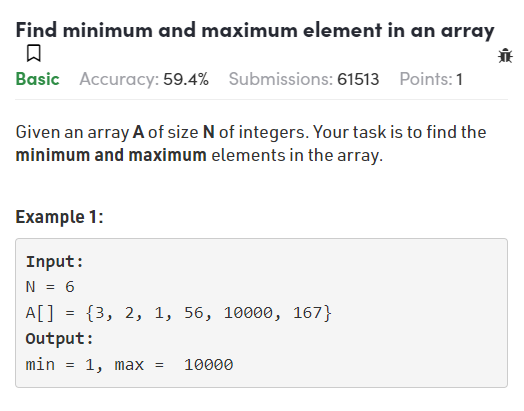
            arr[i] = arr[i-1];

        }

        arr[0] = temp;

    }

}



*/\**

*class pair*

*{*

*long first, second;*

*public pair(long first, long second)*

*{*

*this.first = first;*

*this.second = second;*

*}*

*} \*/*

class Compute

{

    static pair getMinMax(long a[], long n)

    {

        long min = Long.MAX\_VALUE;

        long max = Long.MIN\_VALUE;

        for(int i=0;i<a.length;i++){

            long el = a[i];

            min = Math.min(min,el);

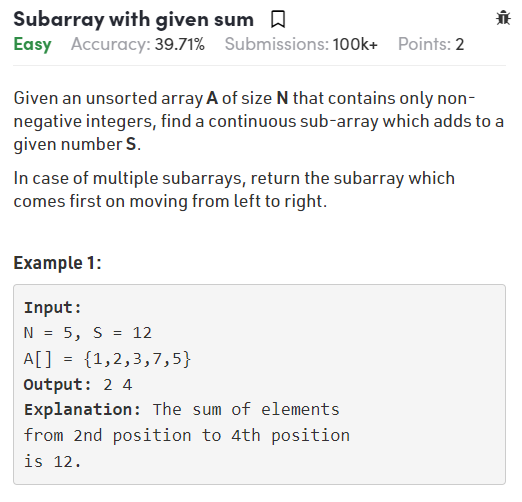
            max = Math.max(max,el);

        }

        return new pair(min,max);

    }

}



class Solution

{

*//Function to find a continuous sub-array which adds up to a given number.*

    static ArrayList<Integer> subarraySum(int[] arr, int n, int s)

    {

       int start = 0;

       int curSum = arr[0];

       ArrayList<Integer> list = new ArrayList<>();

       for(int i=1;i<=arr.length;i++){

*// If curr\_sum exceeds the sum,*

*// then remove the starting elements*

           while(curSum>s && start< i){

               curSum -= arr[start];

               start++;

           }

*// If curr\_sum becomes equal to sum,*

*// then return true*

           if(curSum == s){

               list.add(start+1);

               list.add(i);

                return list;

           }

           if(i<n){

               curSum = curSum + arr[i];

           }

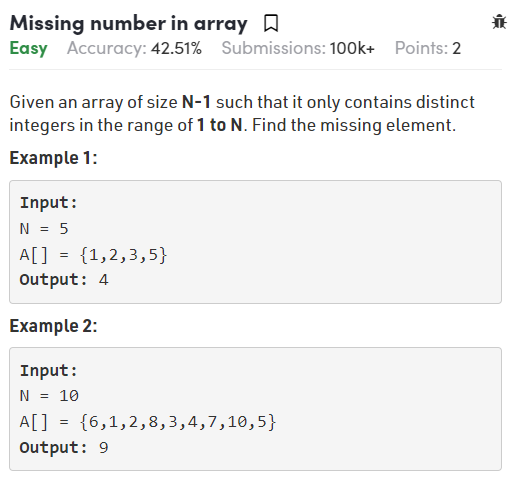
       }

       list.add(-1);

       return list;

    }

}



class Solution {

    int MissingNumber(int array[], int n) {

*// Your Code Here*

        Arrays.sort(array);

        int sum=0;

        for(int i=0;i<array.length;i++){

            sum+=array[i];

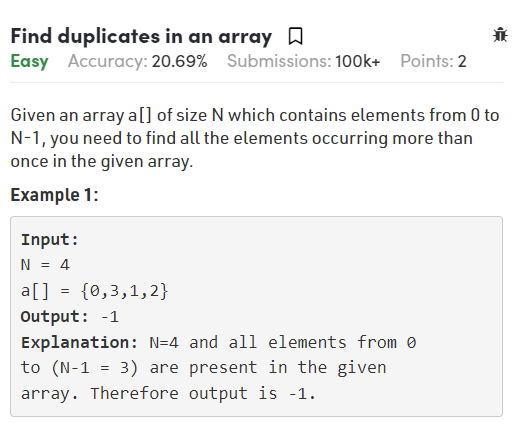
        }

        int total = (n\*(n+1))/2;

        return total-sum;

    }

}



class Solution {

    public static ArrayList<Integer> duplicates(int arr[], int n) {

*// code here*

*// first way passes 303/350 cases*

*/\**

*ArrayList<Integer> list = new ArrayList<>();*

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

*for(int j=i+1;j<n;j++){*

*if(arr[i]==arr[j]  && !list.contains(arr[i])){*

*list.add(arr[i]);*

*}*

*}*

*}*

*if(list.size()==0){*

*list.add(-1);*

*};*

*Collections.sort(list);*

*return list;*

*\*/*

*// second way passes all cases*

        ArrayList<Integer> list = new ArrayList<>();

        HashMap<Integer,Integer> map = new HashMap<>();

        for(int i=0;i<n;i++){

            map.put(arr[i],map.getOrDefault(arr[i],0)+1);

        }

        for(Map.Entry e : map.entrySet()){

            if((Integer.parseInt(e.getValue().toString()))>1){

                list.add(Integer.parseInt(e.getKey().toString()));

            }

        }

        if(list.size()==0){

            list.add(-1);

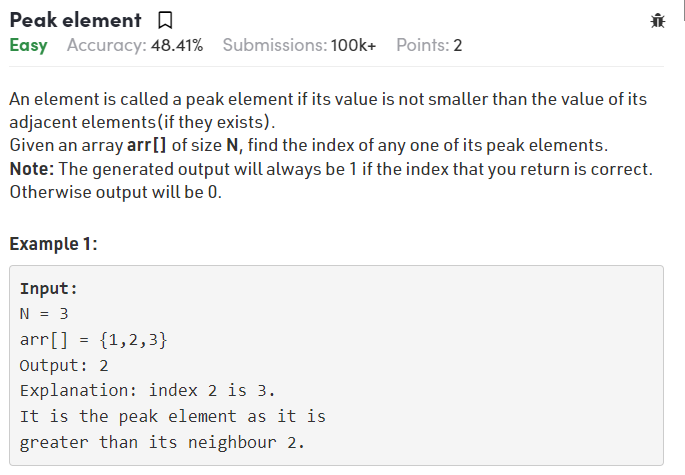
        };

        Collections.sort(list);

        return list;

    }

}



class Solution

{

*// Function to find the peak element*

*// arr[]: input array*

*// n: size of array a[]*

    public int peakElement(int[] arr,int n)

    {

      int start = 0;

        int end = arr.length - 1;

        while (start < end) {

            int mid = start + (end - start) / 2;

            if (arr[mid] > arr[mid+1]) {

*// you are in dec part of array*

*// this may be the ans, but look at left*

*// this is why end != mid - 1*

                end = mid;

            } else {

*// you are in asc part of array*

                start = mid + 1; *// because we know that mid+1 element > mid element*

            }

        }

*// in the end, start == end and pointing to the largest number because of the 2 checks above*

*// start and end are always trying to find max element in the above 2 checks*

*// hence, when they are pointing to just one element, that is the max one because that is what the checks say*

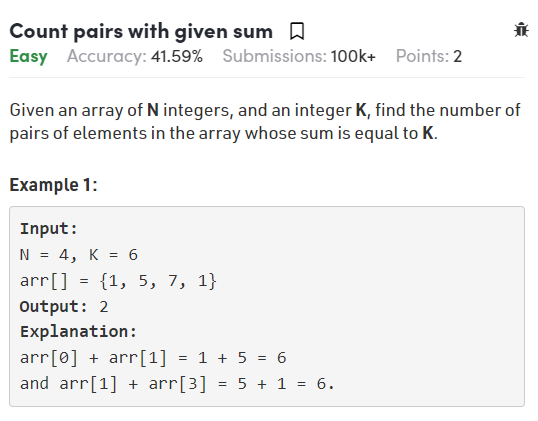
*// more elaboration: at every point of time for start and end, they have the best possible answer till that time*

*// and if we are saying that only one item is remaining, hence cuz of above line that is the best possible ans*

        return start; *// or return end as both are =*

    }

}



**Naive Solution –**A **simple solution** is to traverse each element and check if there’s another number in the array which can be added to it to give sum.

**Efficient solution –**  
A better solution is possible in O(n) time. Below is the Algorithm –

1. Create a map to store frequency of each number in the array. (Single traversal is required)
2. In the next traversal, for every element check if it can be combined with any other element (other than itself!) to give the desired sum. Increment the counter accordingly.
3. After completion of second traversal, we’d have twice the required value stored in counter because every pair is counted two times. Hence divide count by 2 and return.

class Solution {

    int getPairsCount(int[] arr, int n, int k) {

*// first way passes 200/263 testcases rest cases not*

*// executed due to time limit exceed error.*

*// since it uses two loops so time complexity is very large*

*/\**

*int count=0;*

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

*for(int j=i+1;j<n;j++){*

*if(arr[i]+arr[j] == k){*

*count++;*

*}*

*}*

*}*

*return count;*

*\*/*

*// second way , we have to optimize this solution*

        HashMap<Integer,Integer> m = new HashMap<>();

        int count=0;

        for (int i = 0; i < n; i++) {

        if (m.containsKey(k - arr[i])) {

            count += m.get(k - arr[i]);

        }

        if(m.containsKey(arr[i])){

            m.put(arr[i], m.get(arr[i])+1);

        }

        else{

            m.put(arr[i], 1);

             }

         }

        return count;

    }

}

**Wave Array**

**Easy**Accuracy: 62.9% Submissions: 92509 Points: 2

Given a sorted array **arr[]** of distinct integers. Sort the array into a wave-like array and return it  
In other words, arrange the elements into a sequence such that arr[1] >= arr[2] <= arr[3] >= arr[4] <= arr[5].....

If there are multiple solutions, find the lexicographically smallest one.

**Example 1:**

**Input:**

n = 5

arr[] = {1,2,3,4,5}

**Output:** 2 1 4 3 5

**Explanation:** Array elements after

sorting it in wave form are

2 1 4 3 5.

**Example 2:**

**Input:**

n = 6

arr[] = {2,4,7,8,9,10}

**Output:** 4 2 8 7 10 9

**Explanation:** Array elements after

sorting it in wave form are

4 2 8 7 10 9.

**Your Task:**  
The task is to complete the function **convertToWave**(), which converts the given array to a wave array.

public static void swap(int[] arr, int i, int j){

    int temp = arr[i];

    arr[i] = arr[j];

    arr[j] = temp;

}

public static void convertToWave(int arr[], int n){

    Arrays.sort(arr);

*// sort an array*

    for (int i = 0; i < n - 1; i += 2) {

*// swap adjacent elements*

*// 1, 2, 3, 4, 5 => 2, 1, 4, 3, 5*

*// 1 <-> 2,  4 <-> 3  1 swapped with 2 so on....*

*//for this we will start from 0 then jump to second element*

*// coz 0,1 will be swapped.*

        swap(arr, i, i + 1);

    }

}

**Reverse array in groups**

Given an array arr[] of positive integers of size N. Reverse every sub-array group of size K.

**Example 1:**

**Input:**

N = 5, K = 3

arr[] = {1,2,3,4,5}

**Output:** 3 2 1 5 4

**Explanation:** First group consists of elements

1, 2, 3. Second group consists of 4,5.

**Example 2:**

**Input:**

N = 4, K = 3

arr[] = {5,6,8,9}

**Output:** 8 6 5 9

class Solution {

*//Function to reverse every sub-array group of size k.*

    public static void reverse(ArrayList<Integer> arr, int i, int j){

        while(i<j){

            int temp = arr.get(i);

            arr.set(i,arr.get(j));

            arr.set(j,temp);

            i++;

            j--;

        }

    }

    void reverseInGroups(ArrayList<Integer> arr, int n, int k) {

*// since we have to divide in groups*

*// we reverse 0 to k then check again for next iteration*

*// is it lesser then n or not if yes then reverse that part*

*// if not then reverse remaining whole part that's it*

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

*//If (ith index +k) is less than total number of elements it means*

*//k elements exist from current index so we reverse k elements*

*//starting from current index.*

            if(i+k < n){

*//reverse function called to reverse any part of the array.*

                reverse(arr,i,i+k-1);

            }

*//Else k elements from current index doesn't exist.*

*//In that case we just reverse the remaining elements.*

            else{

*//reverse function called to reverse any part of the array.*

                reverse(arr,i,n-1);

            }

        }

    }

}

**Check if two arrays are equal or not**

**Basic**Accuracy: 50.0% Submissions: 39592 Points: 1

Given two arrays **A** and **B** of equal size **N**, the task is to find if given arrays are equal or not. Two arrays are said to be equal if both of them contain same set of elements, arrangements (or permutation) of elements may be different though.  
**Note :** If there are repetitions, then counts of repeated elements must also be same for two array to be equal.

**Example 1:**

**Input:**

N = 5

A[] = {1,2,5,4,0}

B[] = {2,4,5,0,1}

**Output:** 1

**Explanation:** Both the array can be

rearranged to {0,1,2,4,5}

**Example 2:**

**Input:**

N = 3

A[] = {1,2,5}

B[] = {2,4,15}

**Output:** 0

**Explanation:** A[] and B[] have only

one common value.

class Solution{

*//Function to check if two arrays are equal or not.*

    public static boolean check1(long A[],long B[],int N)

    {

*// sort both arrays then order will also be same*

*// then iterate over array anyone element is not same return false;*

        Arrays.sort(A);

        Arrays.sort(B);

        for(int i=0;i<N;i++){

            if(A[i]!=B[i]){

                return false;

            }

        }

        return true;

    }

*//Function to check if two arrays are equal or not.*

    public static boolean check2(long A[],long B[],int N)

    {

*//using a HashMap to store frequency of elements.*

    HashMap<Long, Long> hm = new HashMap<Long, Long>();

*//incrementing frequencies of elements present in first array in HashMap.*

    for(int i = 0;  i< N; i++)

        {

            long num = A[i];

            if(hm.containsKey(num))

            {

                long freq = hm.get(num);

                freq++;

                hm.put(num, freq);

            }

            else{

                hm.put(num, (long)1);

            }

        }

*//decrementing frequencies of elements present in*

*//second array in the HashMap.*

        for(int i = 0; i < N; i++)

        {

            long num = B[i];

            if(hm.containsKey(num))

            {

                long freq = hm.get(num);

                freq--;

                hm.put(num, freq);

            }

        }

*//iterating over the HashMap.*

        for(Map.Entry<Long, Long> entry: hm.entrySet())

        {

*//if frequency of any element in HashMap now is not zero it means*

*//that its count in two arrays was not equal so arrays aren't equal.*

            if(entry.getValue() != 0)

            {

*//returning false since arrays are not equal.*

                return false;

            }

        }

*//returning true if arrays are equal.*

        return true;

    }

}

**Convert array into Zig-Zag fashion**

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.

**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

**Example 2:**

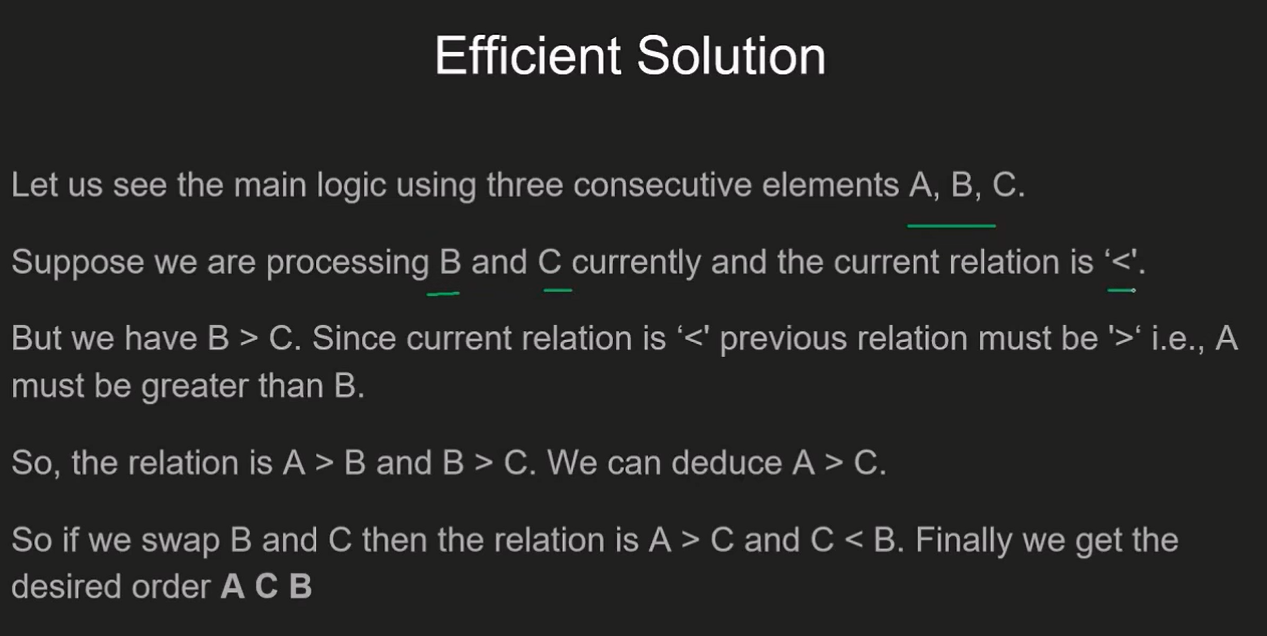
**Input:**

N = 4

Arr[] = {1, 4, 3, 2}

**Output:** 1 4 2 3

**Explanation:** 1 < 4 > 2 < 3



class Solution {

    void swap(int[] arr, int i, int j){

        int temp = arr[i];

    arr[i] = arr[j];

    arr[j] = temp;

}

void zigZag(int arr[], int n) {

*// First Solution is to sort the whole array*

*// and then exclude first one and interchange rest*

*// elements pair wise*

*/\**

*A Simple Solution is to first sort the array. After sorting, exclude the first*

*elemen*

*swap the remaining elements in pairs. (i.e. keep arr[0] as it is, swap arf[1] and*

*arr[2], swap arr[3] and arr[4], and so on). Time complexity is O(nlogn) since we*

*need to sort the array first.*

*\*/*

*/\**

*Arrays.sort(arr);*

*for(int i=1;i<n-1;i+=2){*

*swap(arr,i,i+1);*

*}*

*\*/*

*/\**

*Efficient approach*

*We can convert in O(n) time using an Efficient Approach.*

*The idea is to use modified one pass of bubble sort.*

*Maintain a flag for representing which order(i.e. < or >) currently we need. If the*

*current two elements are not in that order then swap those elements otherwise*

*not.*

*\*/*

    boolean flag = true;

    for (int i = 0; i < n - 1; i++) {

        if (flag) {

            if (arr[i] > arr[i + 1]) {

                swap(arr, i, i + 1);

            }

        } else {

            if (arr[i] < arr[i + 1]) {

                swap(arr, i, i + 1);

            }

        }

        flag = !flag;

    }

}

}

**Minimize the sum of product**

**Basic**Accuracy: 56.38% Submissions: 12565 Points: 1

**[[](https://practice.geeksforgeeks.org/summer-carnival-2022?utm_source=practiceproblems&utm_medium=problemspromobanner&utm_campaign=gsc22) Geeks Summer Carnival is LIVE NOW](https://practice.geeksforgeeks.org/summer-carnival-2022?utm_source=practiceproblems&utm_medium=problemspromobanner&utm_campaign=gsc22" \t "_blank)**

You are given two arrays,**A** and **B**, of equal size **N**. The task is to find the minimum value of **A[0] \* B[0] + A[1] \* B[1] +…+ A[N-1] \* B[N-1],** where shuffling of elements of arrays **A** and **B** is allowed.  
  
  
**Example 1:**

**Input:**

N = 3

A[] = {3, 1, 1}

B[] = {6, 5, 4}

**Output:**

23

**Explanation:**

1\*6+1\*5+3\*4 = 6+5+12

= 23 is the minimum sum

**Example 2:**

**Input:**

N = 5

A[] = {6, 1, 9, 5, 4}

B[] = {3, 4, 8, 2, 4}

**Output:**

80

**Explanation:**

2\*9+3\*6+4\*5+4\*4+8\*1

=18+18+20+16+8

= 80 is the minimum sum

class Solution {

    void reverse(long[] arr){

        int s = 0;

        int e = arr.length-1;

        while(s<=e){

            long temp = arr[s];

            arr[s] = arr[e];

            arr[e] = temp;

            s++;

            e--;

        }

    }

    public long minValue(long a[], long b[], long n)

    {

        // Your code goes here

        Arrays.sort(a);

        Arrays.sort(b);

        reverse(b);

        long prod = 0;

        for(int i=0;i<n;i++){

            prod += a[i]\*b[i];

        }

        return prod;

    }

}

**First element to occur k times**

Given an array of **N** integers. Find the first element that occurs **K** number of times. 

**Example 1:**

**Input :**

N = 7, K = 2

A[] = {1, 7, 4, 3, 4, 8, 7}

**Output :**

4

**Explanation:**

Both 7 and 4 occur 2 times.

But 4 is first that occurs 2 times.

class Solution

{

    public int firstElementKTime(int[] a, int n, int k) {

        HashMap<Integer,Integer> map = new HashMap<>();

        for(int i=0;i<n;i++){

            // first store frequency of each element

            map.put(a[i],map.getOrDefault(a[i],0)+1);

            // then check that current element is equal to k frequency

            // if current count equal to k return it.

            if (map.get(a[i]) == k) {

                return a[i];

            }

        }

        return -1;

    }

}

**Equilibrium Point**

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.

**Note: Retun the index of Equilibrium point. (1-based index)**

**Example 1:**

**Input:**

n = 5

A[] = {1,3,5,2,2}

**Output:** 3

**Explanation:** For second test case

equilibrium point is at position 3

as elements before it (1+3) =

elements after it (2+2).

**Example 2:**

**Input:**

n = 1

A[] = {1}

**Output:** 1

**Explanation:**

Since its the only element hence

its the only equilibrium point.

**Method 1 (Simple but inefficient)**   
Use two loops. Outer loop iterates through all the element and inner loop finds out whether the current index picked by the outer loop is equilibrium index or not. Time complexity of this solution is O(n^2).

**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 following.

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 index.

// update leftsum for next iteration.

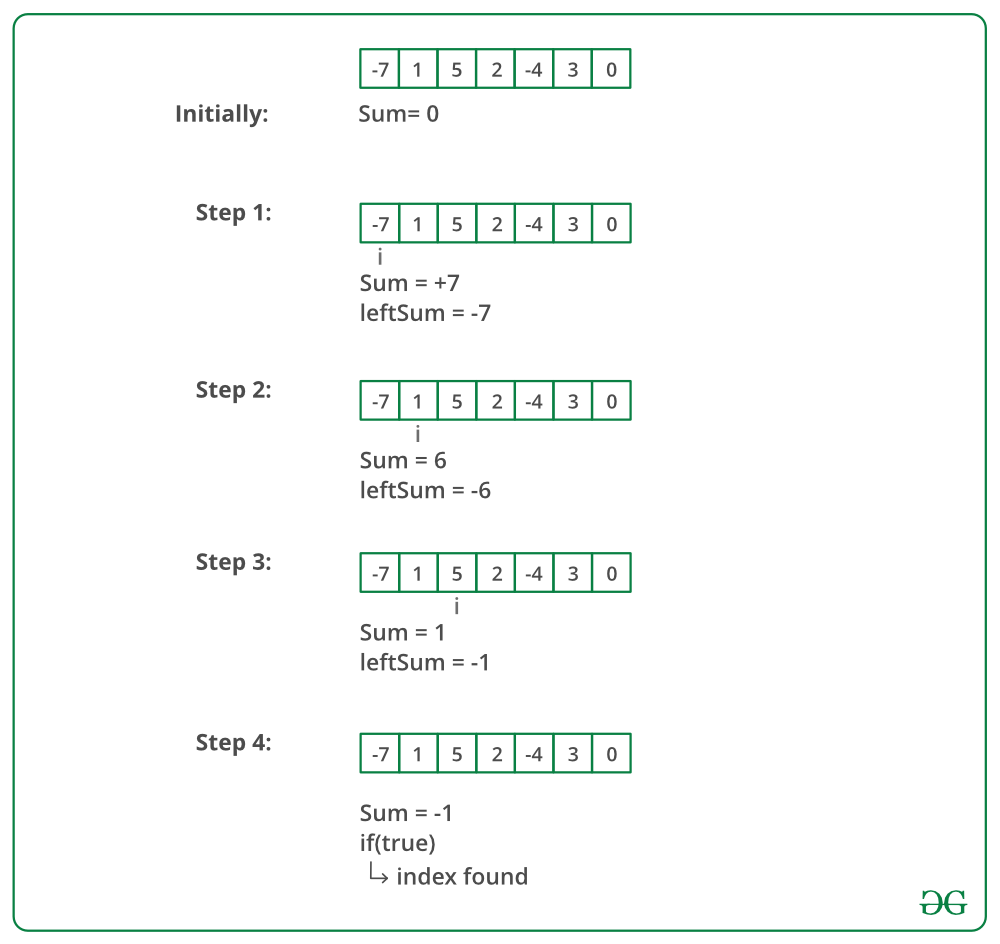
c) leftsum = leftsum + arr[i]

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:



class Solution {

    // a: input array

    // n: size of array

    // Function to find equilibrium point in the array.

    public static long sum(long[] arr,int i,int j){

        long sum=0;

        for(;i<j;i++){

            sum+= arr[i];

        }

        return sum;

    }

    public static int equilibriumPoint(long arr[], int n) {

        // Your code here

        // first way take the sum of left side and right side

        // check both are equal or not but it passes only 200/210 cases

        // since two for loop is used so complexity is higher

        /\*

        for(int i=0;i<n;i++){

            long leftSum = sum(arr,0,i);

            long rightSum = sum(arr,i+1,n);

            if(leftSum == rightSum){

            //       System.out.println(leftSum);

            // System.out.println(rightSum);

                return i+1;

            }

        }

        return -1;

        \*/

        int sum = 0; // initialize sum of whole array

        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 sum for index i

            if (leftsum == sum)

                return i+1;

            leftsum += arr[i];

        }

        /\* If no equilibrium index found, then return 0 \*/

        return -1;

    }

}

**Leaders in an array**

Given an array A of positive integers. Your task is to find the leaders in the array. An element of array is leader if it is greater than or equal to all the elements to its right side. The rightmost element is always a leader.

**Example 1:**

**Input:**

n = 6

A[] = {16,17,4,3,5,2}

**Output:** 17 5 2

**Explanation:** The first leader is 17

as it is greater than all the elements

to its right.  Similarly, the next

leader is 5. The right most element

is always a leader so it is also

included.

**Example 2:**

**Input:**

n = 5

A[] = {1,2,3,4,0}

**Output:** 4 0

class Solution{

    //Function to find the leaders in the array.

    public static boolean isGreatest(int[] arr,int i, int target){

        for(;i<arr.length;i++){

            if(arr[i]>target){

                return false;

            }

        }

        return true;

    }

    static ArrayList<Integer> leaders(int arr[], int n){

        // Your code here

        // using  linear search 231/410 cases passed.

        /\*

        ArrayList<Integer> list = new ArrayList<>();

        for(int i=0;i<n;i++){

            if(isGreatest(arr,i,arr[i])){

                list.add(arr[i]);

            }

        }

        return list;

        \*/

        // scan from right most and add that element to max

        // and it changes when element is greater than max is found

        // while scanning from left to right add all max elements into array

         ArrayList<Integer> list = new ArrayList<>();

         int maxEle =  arr[n-1];

      //We start traversing the array from last element.

        for(int i=n-1; i>=0; i--) {

            //Comparing the current element with the maximum element stored.

            //If current element is greater than max, we add the element.

            if(arr[i] >= maxEle){

                //Updating the maximum element.

                maxEle = arr[i];

                //Storing the current element in arraylist for leaders.

                list.add(maxEle);

            }

        }

         // till now we have whole list of leaders but in reverse order

         // we have to reverse the list

         Collections.reverse(list);

         return list;

    }

}

**Largest subarray with 0 sum**

Given an array having both positive and negative integers. The task is to compute the length of the largest subarray with sum 0.

**Example 1:**

**Input:**

N = 8

A[] = {15,-2,2,-8,1,7,10,23}

**Output:** 5

**Explanation:** The largest subarray with

sum 0 will be -2 2 -8 1 7.

**Efficient Approach:** The brute force solution is calculating the sum of each and every sub-array and checking whether the sum is zero or not. Let's now try to improve the time complexity by taking an extra space of 'n' length. The new array will store the sum of all the elements up to that index. The sum-index pair will be stored in a *hash-map*. A [**Hash map**](https://www.geeksforgeeks.org/java-util-hashmap-in-java/) allows insertion and deletion of key-value pair in constant time. Therefore, the time complexity remains unaffected. So, if the same value appears twice in the array, it will be guaranteed that the particular array will be a zero-sum sub-array. 

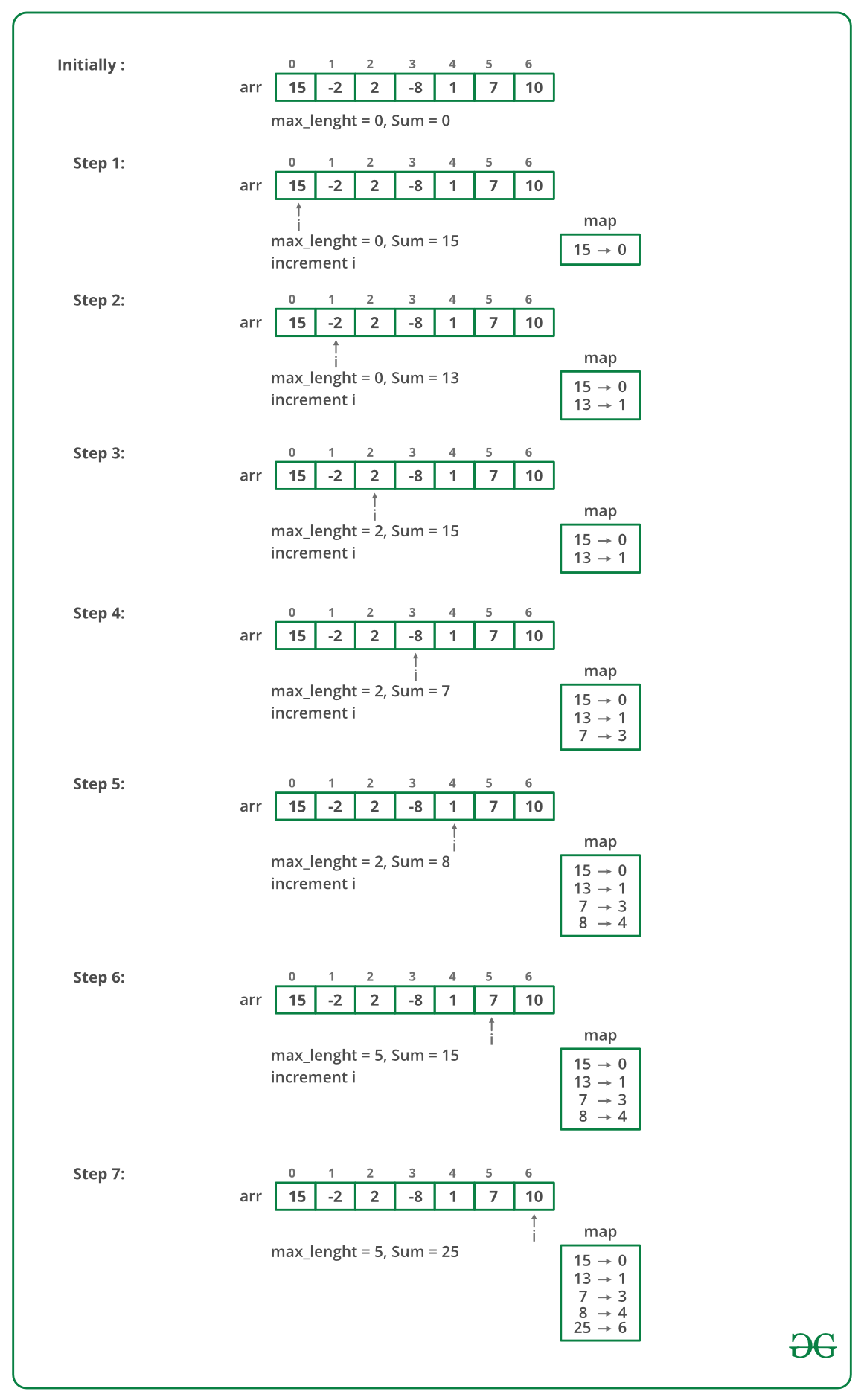
**Mathematical Proof:**

prefix(i) = arr[0] + arr[1] +...+ arr[i]   
prefix(j) = arr[0] + arr[1] +...+ arr[j], j>i   
ifprefix(i) == prefix(j) then prefix(j) - prefix(i) = 0 that means *arr[i+1] + .. + arr[j] = 0*, So a sub-array has zero sum , and the length of that sub-array is j-i+1

**Algorithm:**

1. Create an extra space, an array of *length n (prefix)*, *a variable (sum)*, *length (max\_len)*, and *a hash map (hm)* to store the sum-index pair as a key-value pair.
2. Move along the input array from the start to the end.
3. For every index, update the value of *sum = sum + array[i].*
4. Check every index, if the current sum is present in the hash map or not.
5. If present, update the value of *max\_len* to a maximum difference of two indices (current index and index in the hash-map) and *max\_len.*
6. Else, put the value *(sum)* in the hash map, with the index as a key-value pair.
7. Print the maximum length *(max\_len).*

Below is a dry run of the above approach:



class GfG

{

    int maxLen(int arr[], int n)

    {

      // Creates an empty hashMap hM

        HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();

        int sum = 0; // Initialize sum of elements

        int max\_len = 0; // Initialize result

        // Traverse through the given array

        for (int i = 0; i < arr.length; i++) {

            // Add current element to sum

            sum += arr[i];

            if (arr[i] == 0 && max\_len == 0)

                max\_len = 1;

            if (sum == 0)

                max\_len = i + 1;

            // Look this sum in hash table

            Integer prev\_i = hM.get(sum);

            // If this sum is seen before, then update max\_len

            // if required

            if (prev\_i != null)

                max\_len = Math.max(max\_len, i - prev\_i);

            else // Else put this sum in hash table

                hM.put(sum, i);

        }

        return max\_len;

    }

}

**Array Subset of another array**

Given two arrays: **a1[0..n-1]** of size **n** and **a2[0..m-1]** of size **m**. Task is to check whether a2[] is a subset of a1[] or not. Both the arrays can be sorted or unsorted. It may be assumed that elements in both array are distinct.

**Example 1:**

**Input**:

a1[] = {11, 1, 13, 21, 3, 7}

a2[] = {11, 3, 7, 1}

**Output**:

Yes

**Explanation:**

a2[] is a subset of a1[]

**Example 2:**

**Input**:

a1[] = {1, 2, 3, 4, 5, 6}

a2[] = {1, 2, 4}

**Output**:

Yes

**Explanation:**

a2[] is a subset of a1[]

**Example 3:**

**Input**:

a1[] = {10, 5, 2, 23, 19}

a2[] = {19, 5, 3}

**Output**:

No

**Explanation:**

a2[] is not a subset of a1[]

class Compute {

    public String isSubset( long a1[], long a2[], long n, long m) {

        // sort both arrays

        Arrays.sort(a1);

        Arrays.sort(a2);

        // by default ans is No

        String ans = "No";

        int i=0,j=0;// taking two pointer to point elements in both arrays

        while(i<n && j<m ){// loop

         if( a1[i] == a2[j]){// when both elements equal  increse both

             i++;

             j++;

             continue;

         }else{ // otherwise increment only j

             i++;

         }

        }

         if(j >= m ){ // if j>= m that means all elements of second array is traverssed

         // that means second array is must be a subset of first.

             ans = "Yes";

         }

        return ans;

    }

}

**Auxiliary Space:**O(n)

**Method 5 (Use Set)**

* Insert into the set for the first array; that's how we will know the elements in the array.
* Save the size of the array after inserting the first array element.
* Insert into the same set for the second array.
* Check if the size of the set is still the same or not, if it is then it's true else false.

import java.io.\*;

import java.util.\*;

class GFG

{

  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;

    Set<Integer> s = new HashSet<Integer>();

    for (int i = 0; i < m; i++)

    {

      s.add(arr1[i]);

    }

    int p = s.size();// take previous size

    for (int i = 0; i < n; i++)

    {

      s.add(arr2[i]);

    }

// check that size is incremented or not if incremented than it must not be

// sub set .

    if (s.size() == p)

    {

      System.out.println("arr2[] is subset of arr1[] " + "\n");

    }

    else

    {

      System.out.println("arr2[] is not subset of arr1[] " + "\n" );

    }

  }

}