DSA

**Q1. Peak element**

<https://www.geeksforgeeks.org/problems/peak-element/1?page=1&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=submissions>

Given an *0-indexed* array of integers *arr[] of size n*, find its peak element and return it's index. An element is considered to be peak if it's value is greater than or equal to the values of its adjacent elements (*if they exist*).

*Note: The output will be 1 if the index returned by your function is correct; otherwise, it will be 0.*

**Examples :**

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

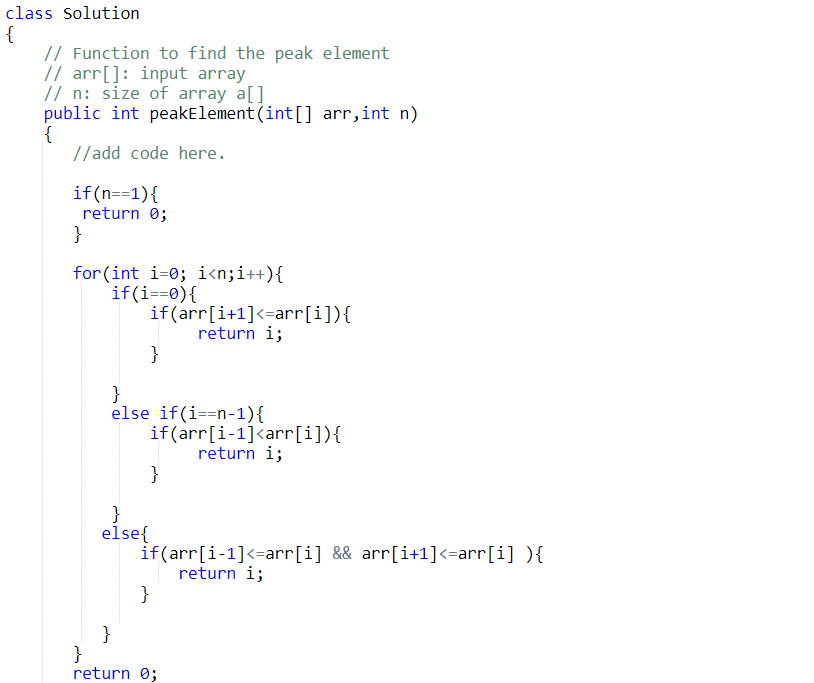
**Output:** 1

**Explanation:** If the index returned is 2, then the output printed will be 1. Since arr[2] = 3 is greater than its adjacent elements, and there is no element after it, we can consider it as a peak element. No other index satisfies the same property, so answer will be printed as 0.

**Input:** n = 7, arr[] = {1, 1, 1, 2, 1, 1, 1}

**Output:** 1

**Explanation:** In this case there are 5 peak elements with indices as {0,1,3,5,6}. Returning any of them will give you correct answer.



**Q2. Array Subset**

[**https://www.geeksforgeeks.org/problems/array-subset-of-another-array2317/1?page=1&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=submissions**](https://www.geeksforgeeks.org/problems/array-subset-of-another-array2317/1?page=1&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=submissions)

**Given two arrays: a1[0..n-1] of size n and a2[0..m-1] of size m, where both arrays may contain duplicate elements. The task is to determine whether array a2 is a subset of array a1. It's important to note that both arrays can be sorted or unsorted. Additionally, each occurrence of a duplicate element within an array is considered as a separate element of the set.**

**Example 1:**

**Input:**

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

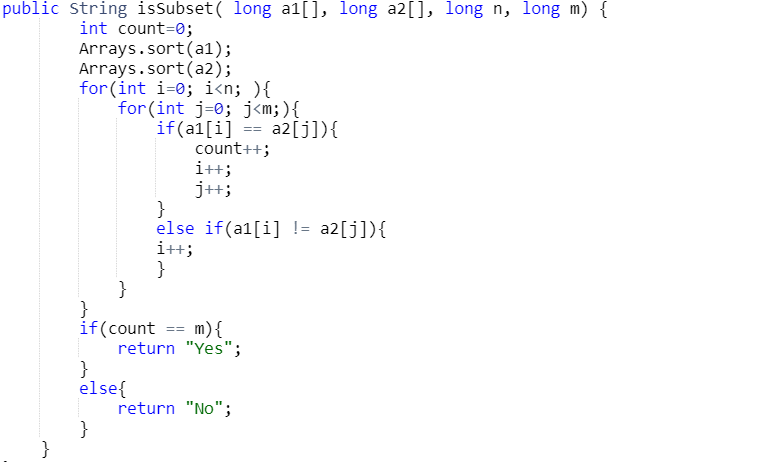
**a2[] = {11, 3, 7, 1, 7}**

**Output:**

**Yes**

**Explanation:**

**a2[] is a subset of a1[]**

****

**class Solution {**

**public boolean isSubset(int a[], int b[]) {**

**int n = a.length;**

**int m = b.length;**

**// Create a hash map to store the frequencies of elements in array a**

**HashMap<Integer, Integer> hm = new HashMap<>();**

**// Increment the frequency of each element in array a**

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

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

**}**

**// Check if each element in array b is present in array a and decrement its**

**// frequency**

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

**if (hm.containsKey(b[i])) {**

**// If frequency becomes 1, remove the element from the hash map**

**if (hm.get(b[i]) == 1) {**

**hm.remove(b[i]);**

**} else {**

**// Decrement the frequency of the element**

**hm.put(b[i], hm.get(b[i]) - 1);**

**}**

**} else {**

**// If an element in array b is not present in array a, return "No"**

**return false;**

**}**

**}**

**// If all elements in array b are present in array a, return "Yes"**

**return true;**

**}**

**}**

**Q3.** **Longest Common Prefix of Strings**

[**https://www.geeksforgeeks.org/problems/longest-common-prefix-in-an-array5129/1?page=1&category=Arrays&sprint=a663236c31453b969852f9ea22507634&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/longest-common-prefix-in-an-array5129/1?page=1&category=Arrays&sprint=a663236c31453b969852f9ea22507634&sortBy=difficulty)

Given an array of strings **arr.** Return the **longest common prefix** among each and every strings present in the array. If there's no prefix common in all the strings, return "**-1**".

**Examples :**

**Input:** arr[] = ["geeksforgeeks", "geeks", "geek", "geezer"]

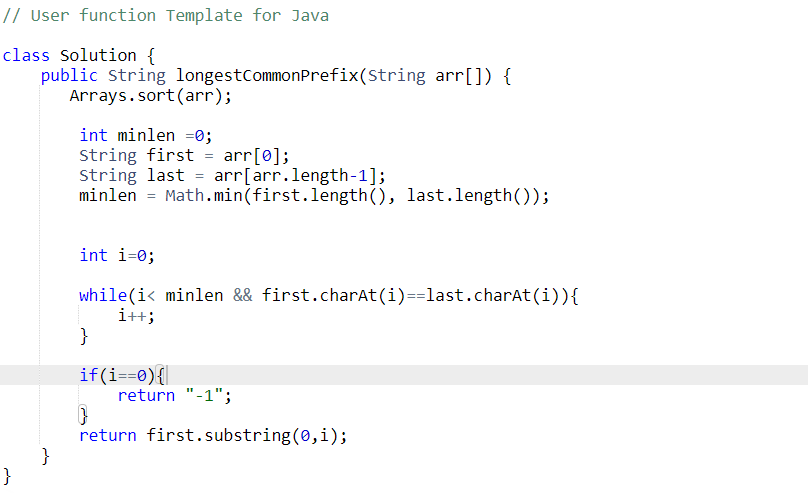
**Output:** gee

**Explanation**: "gee" is the longest common prefix in all the given strings.

**Input**: arr[] = ["hello", "world"]

**Output:** -1

**Explanation**: There's no common prefix in the given strings.



**Q4. Missing in Array**

<https://www.geeksforgeeks.org/problems/missing-number-in-array1416/1?page=1&category=Arrays&sprint=a663236c31453b969852f9ea22507634&sortBy=difficulty>

You are given an array **arr** of size n - 1 that contains distinct integers in the range from 1 to n (inclusive). This array represents a permutation of the integers from 1 to n with one element missing. Your task is to identify and return the missing element.

**Examples:**

**Input:** arr[] = [1,2,3,5]

**Output:** 4

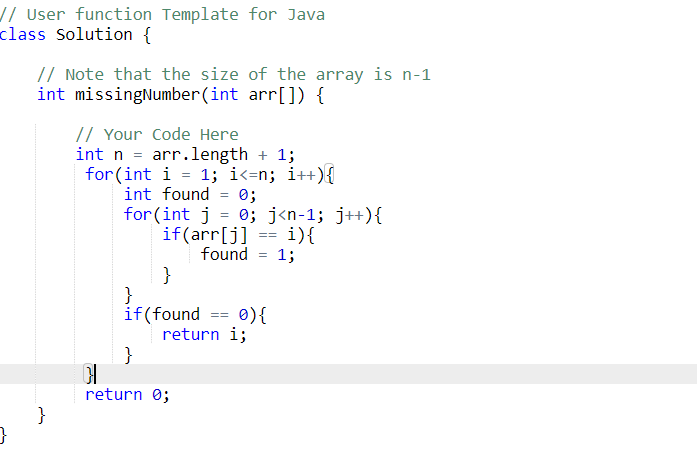
**Explanation :** All the numbers from 1 to 5 are present except 4.

**Input:** arr[] = [1]

**Output:** 2

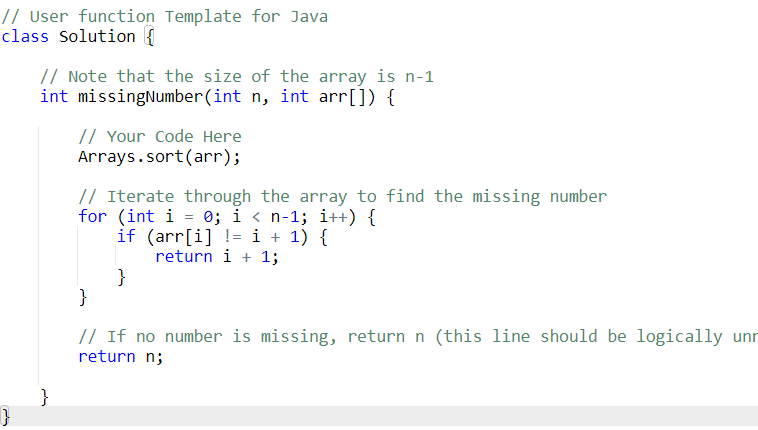
**Explanation:** All the numbers from 1 to 2 are present except 2.

Brute force solution: O(n2)



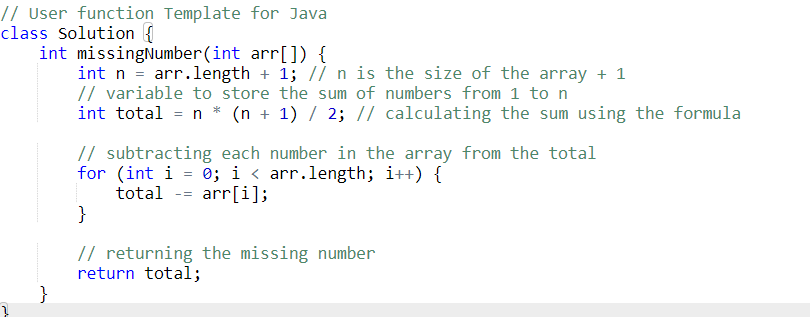
Brute force solution 2: O(nlogn)

As we know that every value in the array is unique so if we sort the array then for every index i in the array its value should be equal to i+1. If it is not then we can surely say that it is our missing number.



Best Solution: o(1)

The summation of the n number is (n\*(n+1)/2) so we calculate this value, and then subtract every value of the array from this and you will get the value of the array which is missing.



**Q5. Sort 0s, 1s and 2s**

[**https://www.geeksforgeeks.org/problems/sort-an-array-of-0s-1s-and-2s4231/1?page=1&category=Arrays&sprint=a663236c31453b969852f9ea22507634&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/sort-an-array-of-0s-1s-and-2s4231/1?page=1&category=Arrays&sprint=a663236c31453b969852f9ea22507634&sortBy=difficulty)

**Given an array arr containing only 0s, 1s, and 2s. Sort the array in ascending order.**

**Examples:**

**Input: arr[] = [0, 2, 1, 2, 0]**

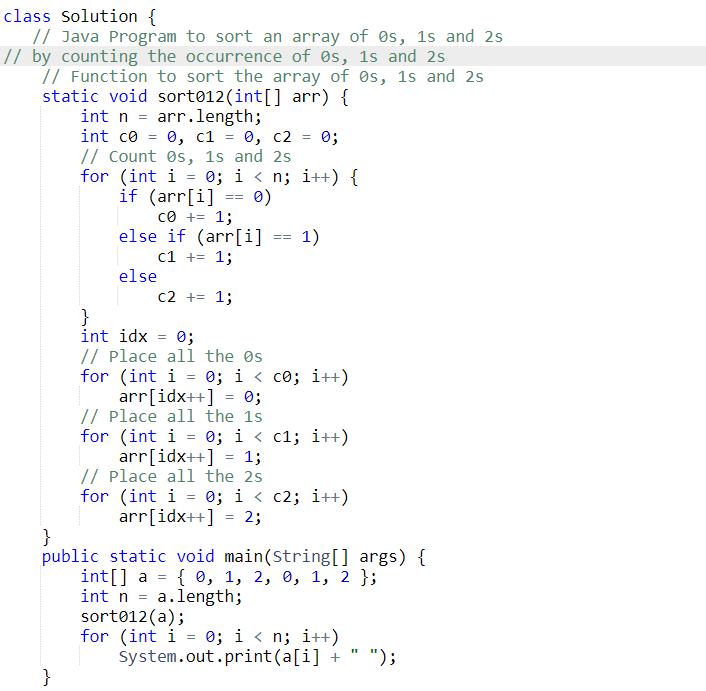
**Output: 0 0 1 2 2**

**Explanation: 0s 1s and 2s are segregated into ascending order.**

**Input: arr[] = [0, 1, 0]**

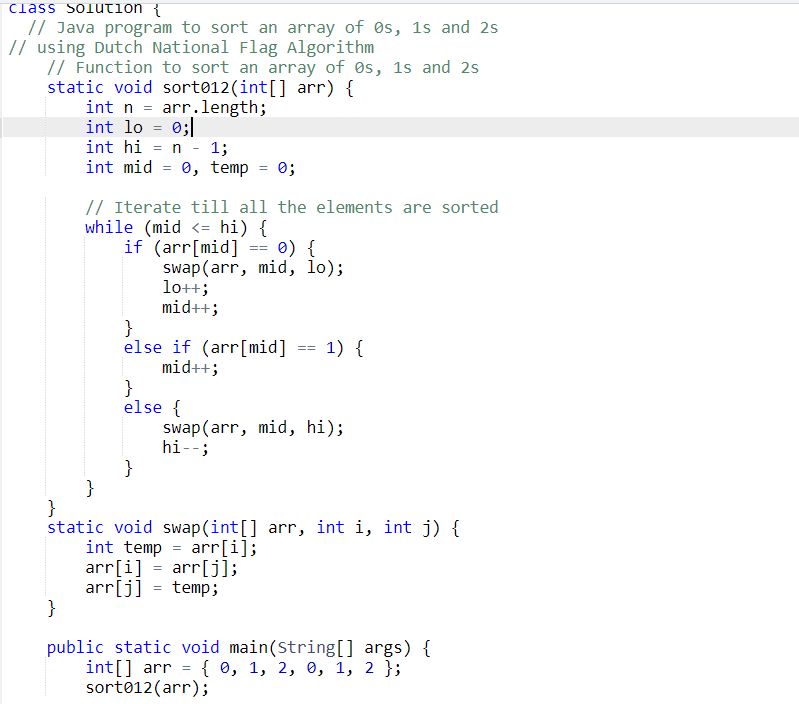
**Output: 0 0 1**

**Explanation: 0s 1s and 2s are segregated into ascending order.**

****

The issues with this approach are:

1. It would not work if 0s and 1s represent keys of objects.
2. Not stable
3. Requires two traversals



**Q6. Rotate Array by One**

[**https://www.geeksforgeeks.org/problems/cyclically-rotate-an-array-by-one2614/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/cyclically-rotate-an-array-by-one2614/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**Given an array arr, rotate the array by one position in clock-wise direction.**

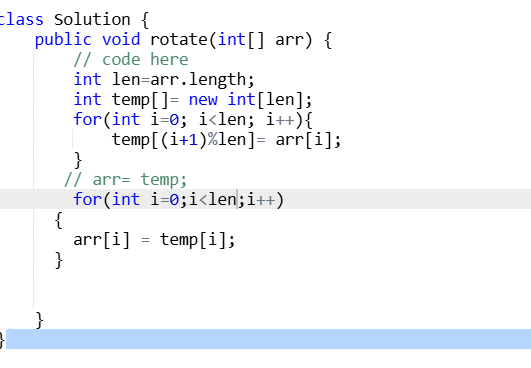
**Examples:**

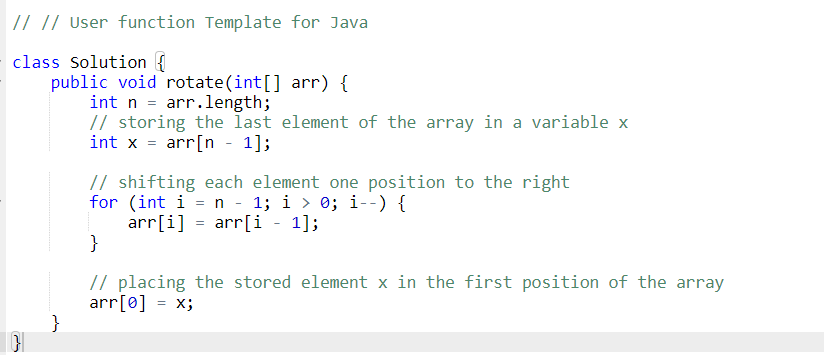
**Input: arr = [1, 2, 3, 4, 5]**

**Output: [5, 1, 2, 3, 4]  
Explanation: If we rotate arr by one position in clockwise 5 come to the front and remaining those are shifted to the end.**

**Input: arr = [9, 8, 7, 6, 4, 2, 1, 3]**

**Output: [3, 9, 8, 7, 6, 4, 2, 1]  
Explanation: After rotating clock-wise 3 comes in first position.**

****



**Q.7 First Repeating Element**

**Given an array arr[], find the first repeating element. The element should occur more than once and the index of its first occurrence should be the smallest.**

**Note:- The position you return should be according to 1-based indexing.**

**Examples:**

**Input: arr[] = [1, 5, 3, 4, 3, 5, 6]**

**Output: 2**

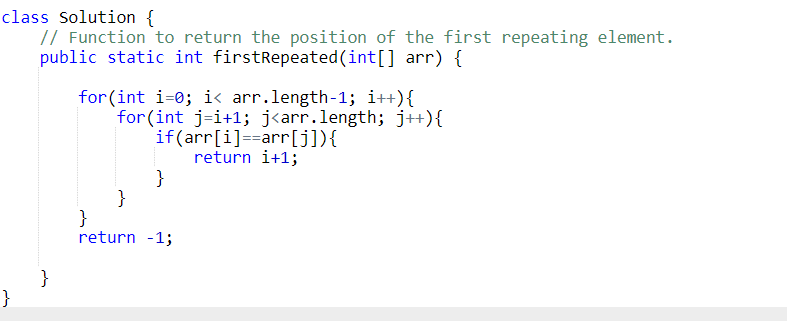
**Explanation: 5 appears twice and its first appearance is at index 2 which is less than 3 whose first the occurring index is 3.**

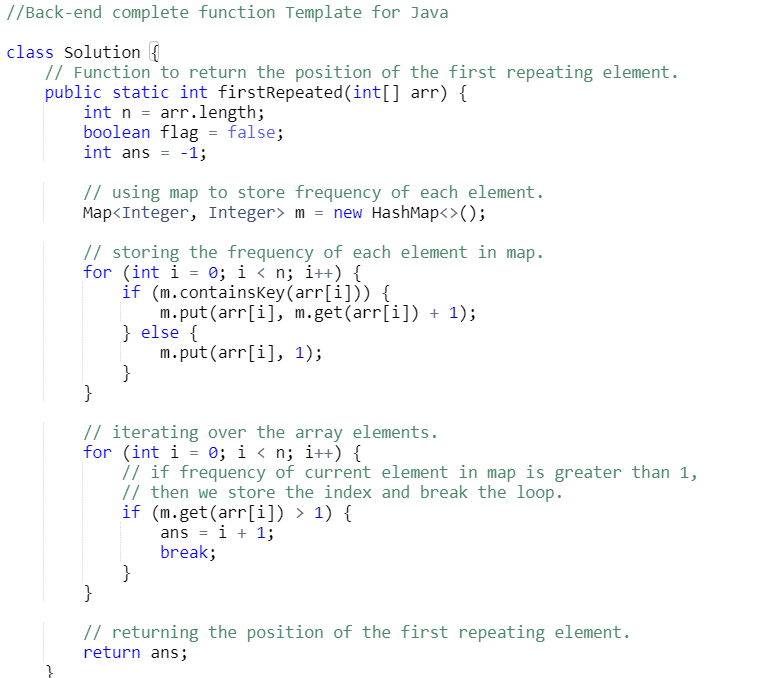
**Input: arr[] = [1, 2, 3, 4]**

**Output: -1**

**Explanation: All elements appear only once so answer is -1.**

<https://www.geeksforgeeks.org/problems/first-repeating-element4018/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty>





**Q8. Sub-arrays with equal number of occurences**

[**https://www.geeksforgeeks.org/problems/sub-arrays-with-equal-number-of-occurences3901/1**](https://www.geeksforgeeks.org/problems/sub-arrays-with-equal-number-of-occurences3901/1)

**Given an array arr[] and two integers say, x and y, find the number of sub-arrays in which the number of occurrences of x is equal to the number of occurrences of y.**

**Examples:**

**Input: arr[] = [1, 2, 1] , x= 1 , y = 2**

**Output: 2**

**Explanation: The possible sub-arrays have same equal number of occurrences of x and y are:**

**1) [1, 2], x and y have same occurrence(1).**

**2) [2, 1], x and y have same occurrence(1).**

**Input: arr[] = [1, 2, 1] , x = 4 , y = 6**

**Output: 6**

**Explanation: The possible sub-arrays have same equal number of occurrences of x and y are:**

**1) [1], x and y have same occurrence(0).**

**2) [2], x and y have same occurrence(0).**

**3) [1], x and y have same occurrence(0).**

**4) [1, 2], x and y have same occurrence(0).**

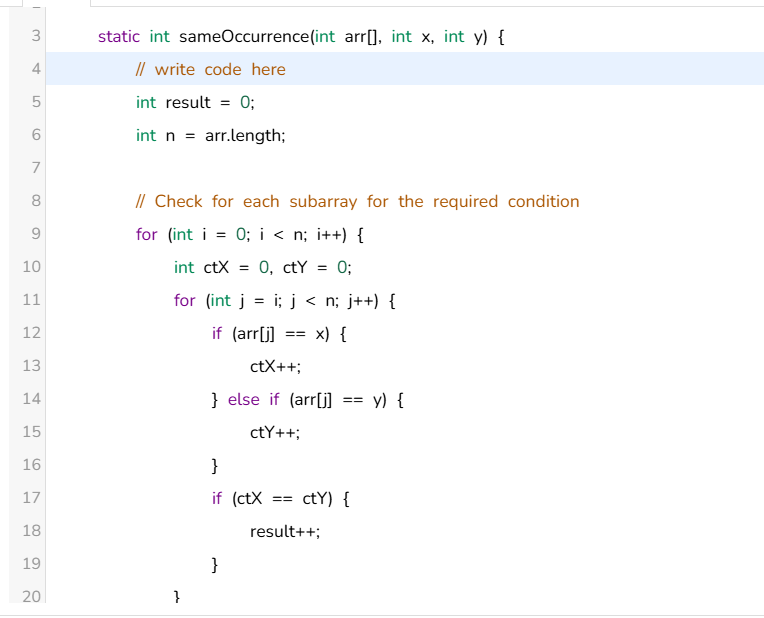
**5) [2, 1], x and y have same occurrence(0).**

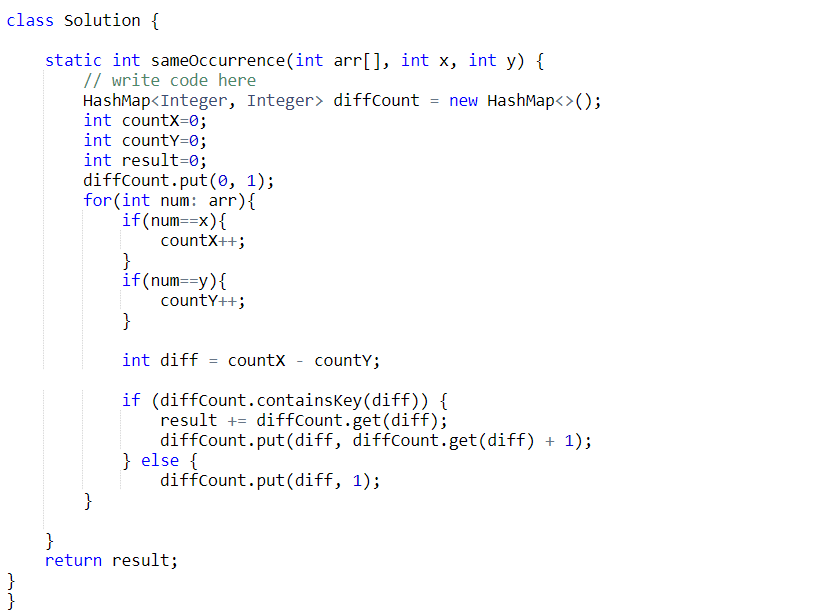
**6) [1, 2, 1], x and y have same occurrence(0).**

**Input: arr[] = [1, 2, 1] , x= 1 , y = 4**

**Output: 1**

**Explanation: The possible sub-array have same equal number of occurrences of x and y is: [2], x and y have same occurrence(0)**

****

****

**Q9. Non-Repeating Element**

<https://www.geeksforgeeks.org/problems/non-repeating-element3958/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty>

Find the first non-repeating element in a given array**arr** of integers and if there is not present any non-repeating element then return **0**

**Note:** The array consists of only positive and negative integers and **not zero**.

**Examples:**

**Input:** arr[] = [-1, 2, -1, 3, 2]

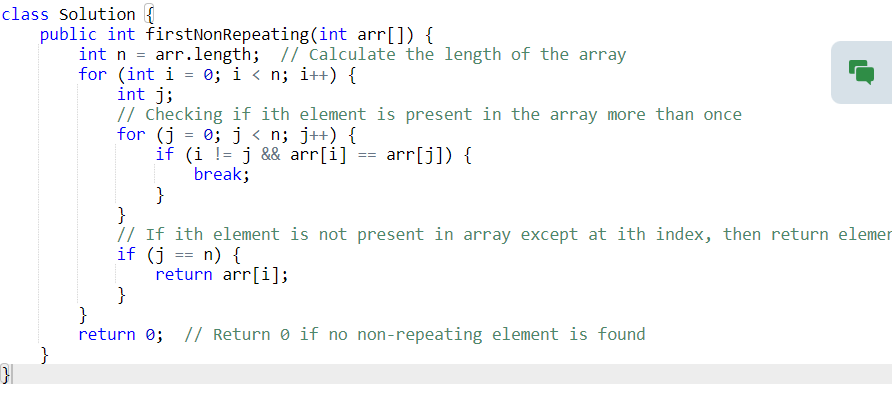
**Output:** 3

**Explanation:** -1 and 2 are repeating whereas 3 is the only number occuring once. Hence, the output is 3.

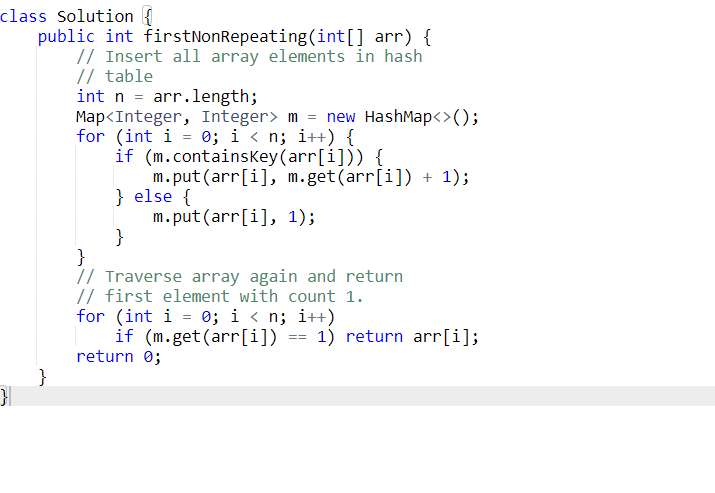
**Input:** arr[] = [1, 1, 1]

**Output:** 0  
**Explanation:** There is not present any non-repeating element so answer should be 0.

**Expected Time Complexity:** O(n).



1. Create an unordered map where keys are the array elements and values are the counts of occurrences of each element.
2. Insert all array elements into the hash table while counting their occurrences.
3. Traverse the array again and return the first element encountered with a count of 1, indicating it's non-repeating.



**Q.10 Move all negative elements to end**

[**https://www.geeksforgeeks.org/problems/move-all-negative-elements-to-end1813/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/move-all-negative-elements-to-end1813/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**Given an unsorted array arr[ ] having both negative and positive integers. The task is to place all negative elements at the end of the array without changing the order of positive elements and negative elements.**

**Note: Don't return any array, just in-place on the array.**

**Examples:**

**Input : arr[] = [1, -1, 3, 2, -7, -5, 11, 6 ]**

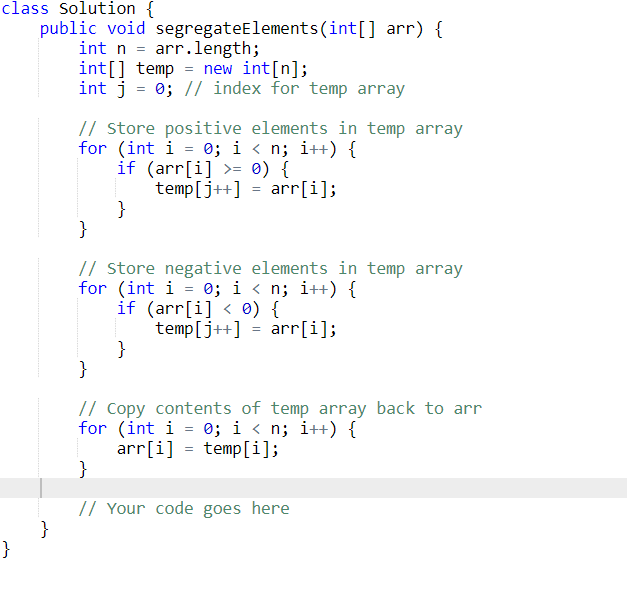
**Output : [1, 3, 2, 11, 6, -1, -7, -5]  
Explanation: By doing operations we separated the integers without changing the order.**

**Input : arr[] = [-5, 7, -3, -4, 9, 10, -1, 11]**

**Output : [7, 9, 10, 11, -5, -3, -4, -1]**

**Expected Time Complexity: O(n)**

1. **Create an empty temporary array temp[] of the same size as the input array.**
2. **Traverse the input array and store positive integers in the temp[] array.**
3. **Check if the input array contains either all positive or all negative integers. If so, return, as there's no need for segregation.**
4. **Store negative integers in the temp[] array after positive integers.**
5. **Copy the contents of the temp[] array back into the original arr[] array using memcpy.**

****

**Q.11 Alternate positive and negative numbers**

[**https://www.geeksforgeeks.org/problems/array-of-alternate-ve-and-ve-nos1401/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/array-of-alternate-ve-and-ve-nos1401/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**Given an unsorted array arr containing both positive and negative numbers. Your task is to rearrange the array and convert it into an array of alternate positive and negative numbers without changing the relative order.  
  
Note:  
- Resulting array should start with a positive integer (0 will also be considered as a positive integer).  
- If any of the positive or negative integers are exhausted, then add the remaining integers in the answer as it is by maintaining the relative order.  
- The array may or may not have equal number of positive and negative integers.**

**Examples:**

**Input: arr[] = [9, 4, -2, -1, 5, 0, -5, -3, 2]**

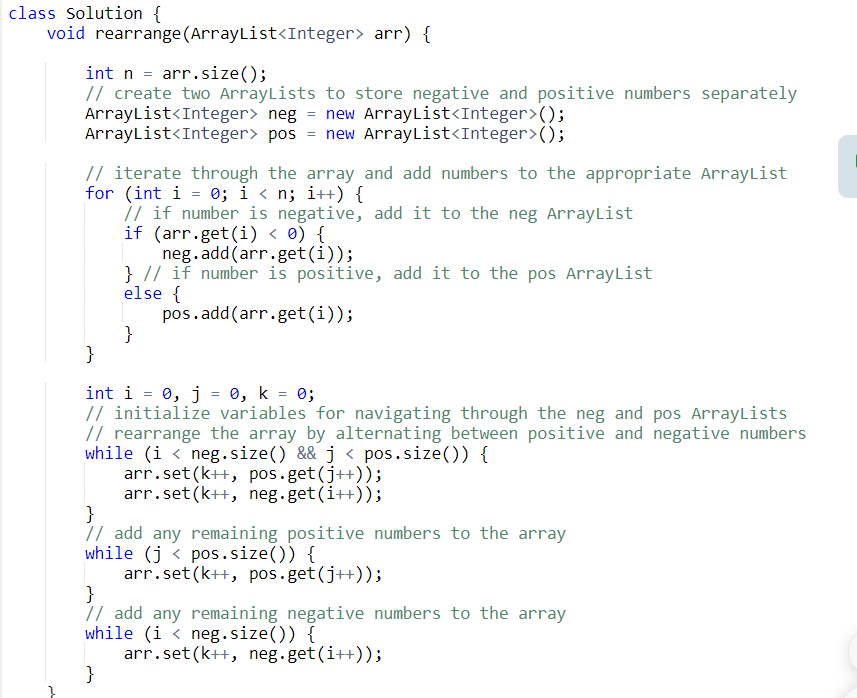
**Output: 9 -2 4 -1 5 -5 0 -3 2**

**Explanation: The positive numbers are [9, 4, 5, 0, 2] and the negative integers are [-2, -1, -5, -3]. Since, we need to start with the positive integer first  
and then negative integer and so on (by maintaining the relative order as well), hence we will take 9 from the positive set of elements and then  
-2 after that 4 and then -1 and so on.  
Hence, the output is 9, -2, 4, -1, 5, -5, 0, -3, 2.**

**Input: arr[] = [-5, -2, 5, 2, 4, 7, 1, 8, 0, -8]**

**Output: 5 -5 2 -2 4 -8 7 1 8 0**

**Explanation : The positive numbers are [5, 2, 4, 7, 1, 8, 0] and the negative integers are [-5,-2,-8]. According to the given conditions we will start  
from the positive integer 5 and then -5 and so on. After reaching -8 there are no negative elements left, so according to the given rule, we will  
add the remaining elements (in this case positive elements are remaining) as it in by maintaining the relative order.  
Hence, the output is 5, -5, 2, -2, 4, -8, 7, 1, 8, 0.**

****

**Q.12 Sum Pair closest to k**

[**https://www.geeksforgeeks.org/problems/pair-in-array-whose-sum-is-closest-to-x1124/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/pair-in-array-whose-sum-is-closest-to-x1124/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**Given a sorted array arr[]  and a number k, find a pair in sorted order in an array whose sum is closest to k.  
Note: If there are multiple such pairs return the pair with maximum absolute difference.**

**Examples:**

**Input: arr[] = [10, 22, 28, 29, 30, 40], k = 54**

**Output: [22, 30]**

**Explanation: As 22 + 30 = 52 is closest to 54.**

**Input: arr[] = [1, 2, 3, 4, 5], k = 10**

**Output: [4, 5]**

**Explanation: As 4 + 5 = 9 is closest to 10.**

**Input: arr[] = [1, 2, 4, 5, 7], k = 10**

**Output: [2, 7]**

**Explanation: As [4,7] and [2,7] both equals 11 and closest to k=10,so abs(7-2)=5 and abs(7-4)=3.Hence,[2,7] has maximum absolute difference and closest to k**

**A simple solution is to consider every pair and keep track of the closest pair (the absolute difference between pair sum and x is minimum). Finally, print the closest pair. The time complexity of this solution is O(n2)**

**An efficient solution can find the pair in O(n) time. The idea is similar to method 1 of**[**this**](https://www.geeksforgeeks.org/write-a-c-program-that-given-a-set-a-of-n-numbers-and-another-number-x-determines-whether-or-not-there-exist-two-elements-in-s-whose-sum-is-exactly-x/)**post. The following is a 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**

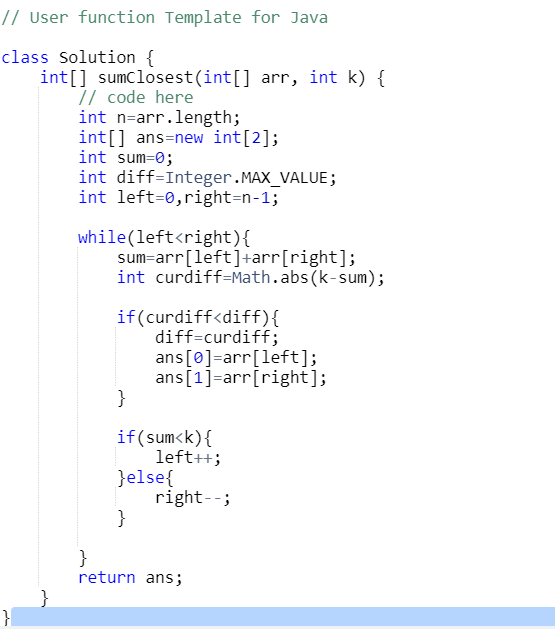
**3) Loop while l < r.**

**(a) If abs(arr[l] + arr[r] - sum) < diff then**

**update diff and result**

**(b) If(arr[l] + arr[r] < sum ) then l++**

**(c) Else r--**

****

**Q.13 Wave Array**

[**https://www.geeksforgeeks.org/problems/wave-array-1587115621/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/wave-array-1587115621/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**Given a sorted array arr[] of distinct integers. Sort the array into a wave-like array(In Place). 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.**

**Note: The given array is sorted in ascending order, and you don't need to return anything to change the original array.**

**Examples:**

**Input: n = 5, arr[] = {1,2,3,4,5}**

**Output: 2 1 4 3 5**

**Explanation: Array elements after sorting it in the waveform are 2 1 4 3 5.**

**Input: n = 6, arr[] = {2,4,7,8,9,10}**

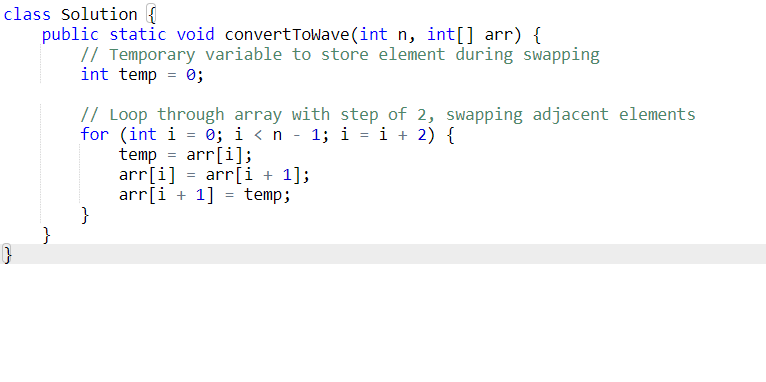
**Output: 4 2 8 7 10 9**

**Explanation: Array elements after sorting it in the waveform are 4 2 8 7 10 9.**

**A Simple Solution is to use sorting. First sort the input array, then swap all adjacent elements.  
As in question input array is already sorted so just we need to swap all adjacent elements to make it a wave array.**

**Example:**

**Input:  
n = 6  
arr[] = {2,4,7,8,9,10}  
swap arr[0] and arr[1]  
swap arr[2] and arr[3]  
swap arr[4] and arr[5]  
Output: 4 2 8 7 10 9**

****

**Alternate Approach:**

**Intuition:**

**The idea is based on the fact that if we make sure that all even positioned (at index 0, 2, 4, ..) elements are greater than their adjacent odd elements, we don't need to worry about oddly positioned elements.**

**Implementation:**

1. **Traverse all even positioned elements of the input array, and do the following.**
   1. **If the current element is smaller than the previous odd element, swap the previous and current.**
   2. **If the current element is smaller than the next odd element, swap the next and current.**

**Q. 14 Array Duplicates**

[**https://www.geeksforgeeks.org/problems/find-duplicates-in-an-array/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/find-duplicates-in-an-array/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**Given an array arr of integers, find all the elements that occur more than once in the array. Return the result in ascending order. If no element repeats, return an empty list.**

**Examples:**

**Input: arr[] = [2, 3, 1, 2, 3]**

**Output: [2, 3]**

**Explanation: 2 and 3 occur more than once in the given array.**

**Input: arr[] = [0, 3, 1, 2]   
Output: []   
Explanation: There is no repeating element in the array, so the output is empty.**

**Input: arr[] = [2]**

**Output: []**

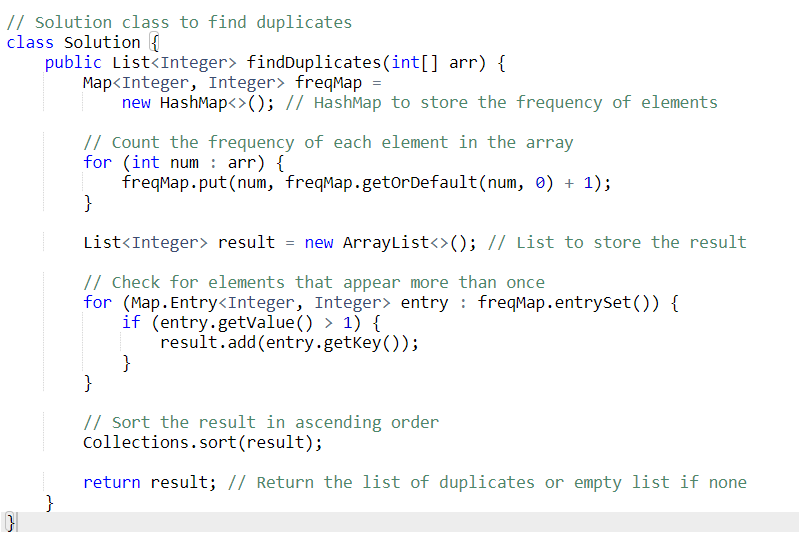
**Explanation: There is single element in the array. Therefore output is empty.**

**1. Iterate through the array and update elements: Initially, iterate through the input array. For each element, increment the value at the index equal to the element modulo the array size by the array size.**

**2. Identify elements occurring more than once: After updating all elements, loop through the array again. Check if the value at index `i` divided by the array size is greater than 1. If it is, the element `i` occurs more than once, hence add it to the output vector.**

**3. Handle cases when no duplicates are found: If no duplicates are detected during the iteration, append `-1` to the output vector.**

**4. Return the final output: Return the vector containing duplicate elements or return an empty vector if no duplicates were found.**



**Q.15 Kth Smallest**

[**https://www.geeksforgeeks.org/problems/kth-smallest-element5635/1?page=2&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/kth-smallest-element5635/1?page=2&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**Given an array arr[] and an integer k where k is smaller than the size of the array, the task is to find the kth smallest element in the given array.**

**Follow up: Don't solve it using the inbuilt sort function.**

**Examples :**

**Input: arr[] = [7, 10, 4, 3, 20, 15], k = 3**

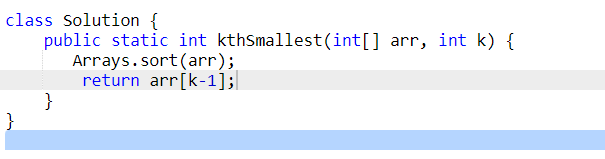
**Output: 7**

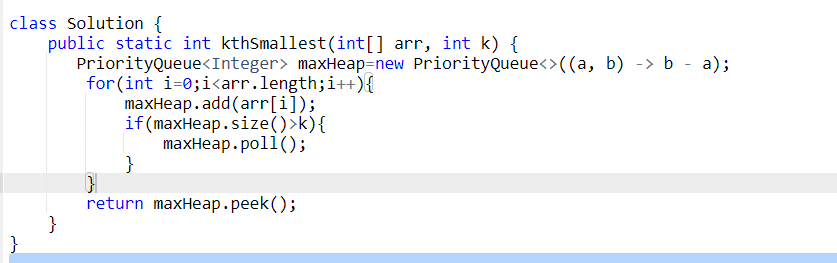
**Explanation: 3rd smallest element in the given array is 7.**

**Input: arr[] = [2, 3, 1, 20, 15], k = 4**

**Output: 15**

**Explanation: 4th smallest element in the given array is 15**





**Expected Approach**

**Intuition**

Idea is to use quick select algorithm which uses a random partition and recursion to efficiently find the kth smallest element in an array. The pivot position is used to determine whether to search in the left or right half of the array.

**Dry Run:**

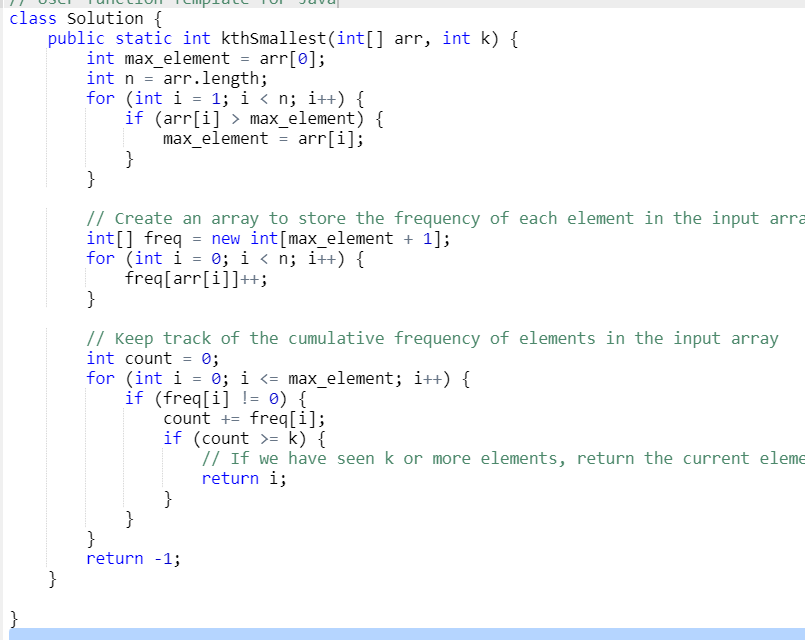
arr = [10, 4, 5, 8, 6, 11, 26]

k = 3

1. **First Call**: kthSmallest(arr, 0, 6, 3)
   * **randomPartition(arr, 0, 6)** is called:
     + The pivot element is randomly selected. Suppose pivot = 4.
     + Swap arr[l + pivot] with arr[r]: Swap arr[0 + 4] (which is 6) with arr[6] (which is 26).
     + **Array after swap**: [10, 4, 5, 8, 26, 11, 6]
     + Now, call partition(arr, 0, 6).
2. **First Call to partition(arr, 0, 6)**:
   * **Pivot**: x = arr[r] = 6
   * **Initial i = 0**
   * Iterate through j from 0 to 5:
     + j=0: arr[j]=10, no swap as arr[j] > pivot.
     + j=1: arr[j]=4, swap arr[i] with arr[j]. Now i=1, arr=[4, 10, 5, 8, 26, 11, 6].
     + j=2: arr[j]=5, swap arr[i] with arr[j]. Now i=2, arr=[4, 5, 10, 8, 26, 11, 6].
     + j=3: arr[j]=8, no swap as arr[j] > pivot.
     + j=4: arr[j]=26, no swap as arr[j] > pivot.
     + j=5: arr[j]=11, no swap as arr[j] > pivot.
   * **Final Swap**: Swap arr[i] with arr[r]. Now arr=[4, 5, 6, 8, 26, 11, 10], i=2.
   * **Return i=2** as the position of the pivot.
3. **Back to kthSmallest(arr, 0, 6, 3)**
   * pos = 2
   * Check if pos-l == k-1:
     + pos-l = 2-0 = 2
     + k-1 = 3-1 = 2
   * **Condition satisfied**: The pivot position is the 3rd smallest element, so return arr[pos] = 6.

**Result**

The 3rd smallest element in the array [10, 4, 5, 8, 6, 11, 26] is 6.



**Q.16 Implement two stacks in an array**

[**https://www.geeksforgeeks.org/problems/implement-two-stacks-in-an-array/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/implement-two-stacks-in-an-array/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**Your task is to implement  2 stacks in one array efficiently. You need to implement 4 methods.**

**twoStacks : Initialize the data structures and variables to be used to implement  2 stacks in one array.  
push1 : pushes element into first stack.  
push2 : pushes element into second stack.  
pop1 : pops element from first stack and returns the popped element. If first stack is empty, it should return -1.  
pop2 : pops element from second stack and returns the popped element. If second stack is empty, it should return -1.**

class twoStacks {

int arr[];

int size;

int top1, top2;

twoStacks() {

size = 100;

arr = new int[100];

top1 = -1;

top2 = size;

}

// Function to push an integer into the stack1.

void push1(int x) {

// if there is space between the top of both stacks

// we push the element at top1+1.

if (top1 < top2 - 1) {

top1++;

arr[top1] = x;

}

}

// Function to push an integer into the stack2.

void push2(int x) {

// if there is space between the top of both stacks

// we push the element at top2-1.

if (top1 < top2 - 1) {

top2--;

arr[top2] = x;

}

}

// Function to remove an element from top of the stack1.

int pop1() {

// if top1==-1, stack1 is empty so we return -1 else we

// return the top element of stack1.

if (top1 >= 0) {

int x = arr[top1];

top1--;

return x;

} else

return -1;

}

// Function to remove an element from top of the stack2.

int pop2() {

// if top2==size of array, stack2 is empty so we return -1 else we

// return the top element of stack2.

if (top2 < size) {

int x = arr[top2];

top2++;

return x;

} else

return -1;

}

}

**Q.17 Max sum path in two arrays**

**Given two sorted arrays of distinct integers arr1 and arr2. Each array may have some elements in common with the other array. Find the maximum sum of a path from the beginning of any array to the end of any array. You can switch from one array to another array only at the common elements.**

**Note:  When we switch from one array to other,  we need to consider the common element only once in the result.**[**https://www.geeksforgeeks.org/problems/max-sum-path-in-two-arrays/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/max-sum-path-in-two-arrays/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**Examples :**

**Input: arr1 = [2, 3, 7, 10, 12] , arr2 = [1, 5, 7, 8]**

**Output: 35**

**Explanation: The path will be 1+5+7+10+12 = 35, where 1 and 5 come from arr2 and then 7 is common so we switch to arr1 and add 10 and 12.**

**Input: arr1 = [1, 2, 3] , arr2[] = [3, 4, 5]**

**Output: 15**

**Explanation: The path will be 1+2+3+4+5=15.**

**Expected Time Complexity: O(m + n)  
Expected Auxiliary Space: O(1)**

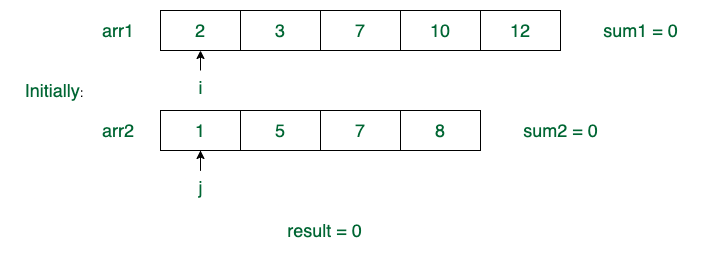
The idea is to do something similar to the merge process of [merge sort](http://geeksquiz.com/merge-sort/). This involves calculating the sum of elements between all common points of both arrays. Whenever there is a common point, compare the two sums and add the maximum of two to the result.

**Illustration:**

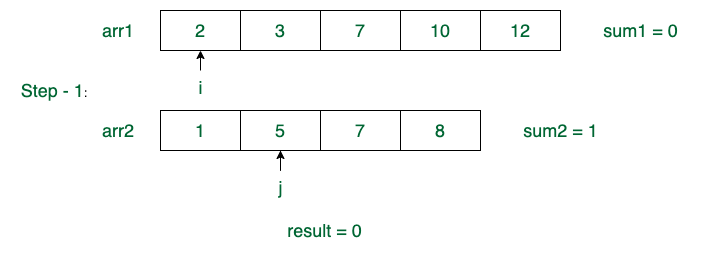
**Input:**arr1[] = {2, 3, 7, 10, 12}

           arr2[] = {1, 5, 7, 8}

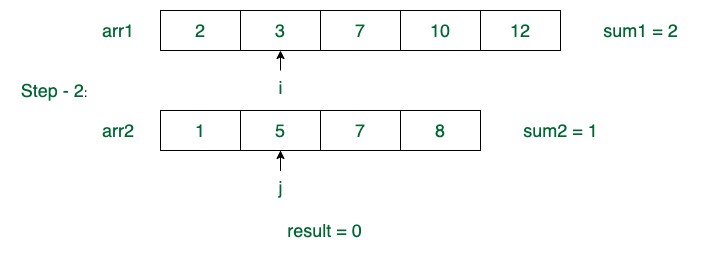
          Intialise i = 0, j = 0, sum1 = 0, sum2 = 0, *result* = 0



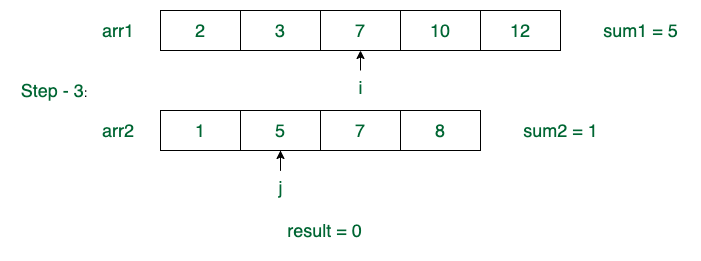
**Step - 1:** arr1[i] > arr2[j]   
sum2 = sum2 + arr2[j] = 0 + 1 = 1  
j = j + 1 = 1



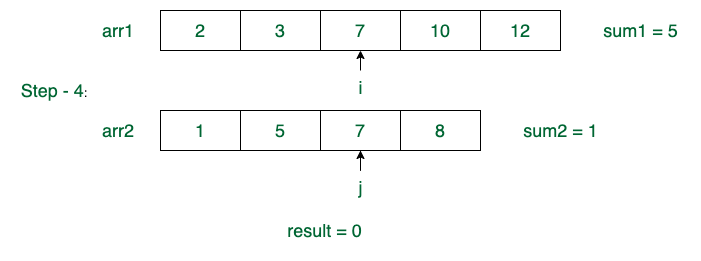
**Step - 2:** arr1[i] < arr2[j]   
sum1 = sum1 + arr1[i] = 0 + 2 = 2  
i = i + 1 = 1



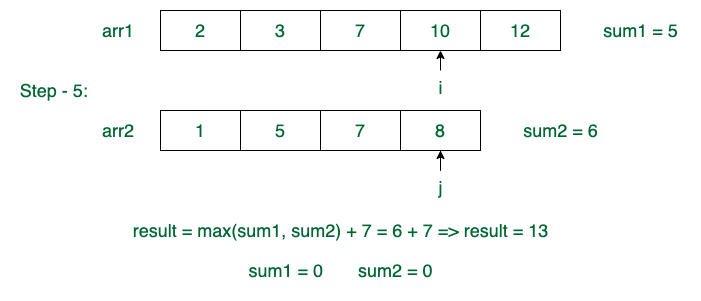
**Step - 3:** arr1[i] < arr2[j]   
sum1 = sum1 + arr1[i] = 2 + 3 = 5  
i = i + 1 = 2



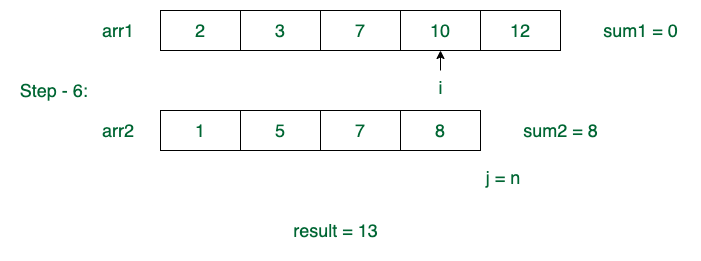
**Step - 4:** arr1[i] > arr2[j]   
sum2 = sum2 + arr2[j] = 1 + 5 = 6  
j = j + 1 = 2



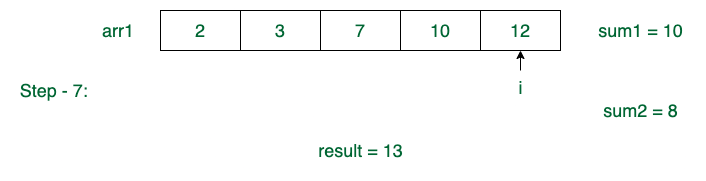
**Step - 5:** arr1[i] == arr2[j]   
*result* = *result* + maximum(sum1, sum2) + arr1[i] = 0 + max(5, 6) + 7 = 13  
sum1 = 0, sum2 = 0  
i = i + 1 = 3  
j = j + 1 = 3



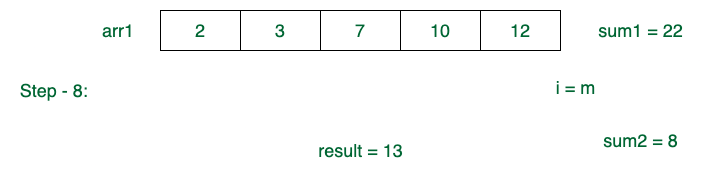
**Step - 6:** arr1[i] > arr2[j]   
sum2 = sum2 + arr2[j] = 0 + 8 = 8  
j = j + 1 = 4



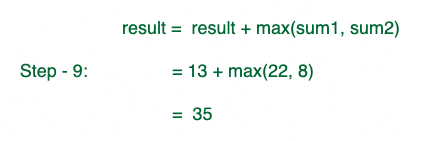
**Step - 7:** sum1 = sum1 + arr1[i] = 0 + 10 = 10  
i = i + 1 = 4



**Step - 8:** sum1 = sum1 + arr1[i] = 10 + 12 = 22  
i = i + 1 = 5



**Step - 9:** *result* = *result* + max(sum1, sum2) = 13 + max(10, 22) = 35



Hence, maximum sum path is **35**.

Follow the steps below to solve the given problem:

* Initialize variables, *result*, *sum1*, *sum2*. Initialize result as 0. Also initialize two variables sum1 and sum2 as 0. Here sum1 and sum2 are used to store sum of element in ar1[] and ar2[] respectively. These sums are between two common points.
* Now run a loop to traverse elements of both arrays. While traversing compare current elements of array 1 and array 2 in the following order.  
  + If current element of *array 1* is smaller than current element of *array 2*, then update *sum1*, else if current element of *array 2* is smaller, then update *sum2*.
  + If the current element of *array 1* and *array 2*are same, then take the maximum of sum1 and sum2 and add it to the result. Also add the common element to the result.
  + This step can be compared to the merging of two *sorted* arrays, If the smallest element of the two current array indices is processed then it is guaranteed that if there is any common element it will be processed together. So the sum of elements between two common elements can be processed

int maxPathSum(vector<int> &arr1, vector<int> &arr2) {

// Code here

int n=arr1.size() , m=arr2.size() ;

int i=0,j=0,sum1=0,sum2=0 ;

arr1.push\_back(INT\_MAX);

arr2.push\_back(INT\_MAX);

while( (i<n || j<m) )

{

if( arr1[i] < arr2[j] )

{

sum1+=arr1[i];

i++;

}

else if( arr1[i] > arr2[j] )

{

sum2+= arr2[j];

j++;

}

else

{

sum1=max(sum1,sum2)+arr1[i];

sum2=sum1;

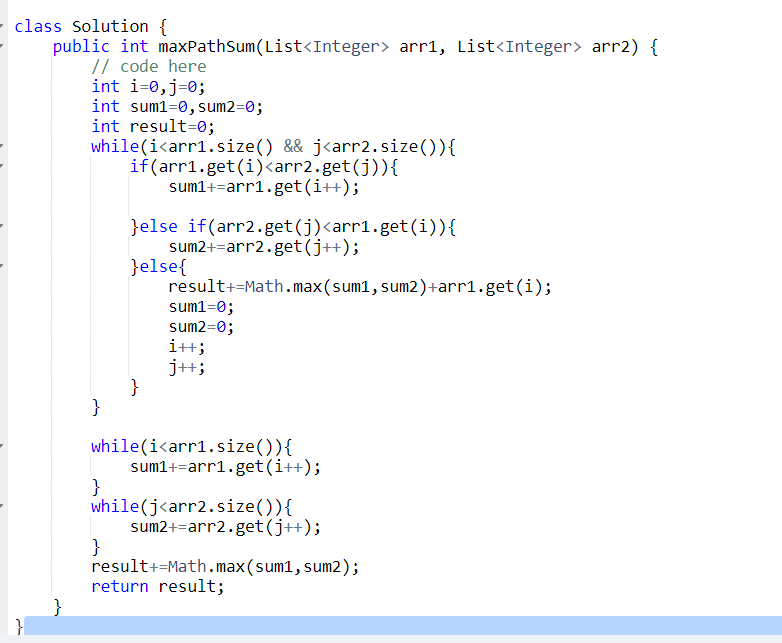
i++,j++;

}

}

return max( sum1 , sum2 );

}



**Q.18 Row with max 1s**

[**https://www.geeksforgeeks.org/problems/row-with-max-1s0023/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/row-with-max-1s0023/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**You are given a 2D array consisting of only 1's and 0's, where each row is sorted in non-decreasing order. You need to find and return the index of the first row that has the most number of 1s. If no such row exists, return -1.  
Note: 0-based indexing is followed.**

**Examples:**

**Input: arr[][] = [[0, 1, 1, 1],  
 [0, 0, 1, 1],  
 [1, 1, 1, 1],  
 [0, 0, 0, 0]]**

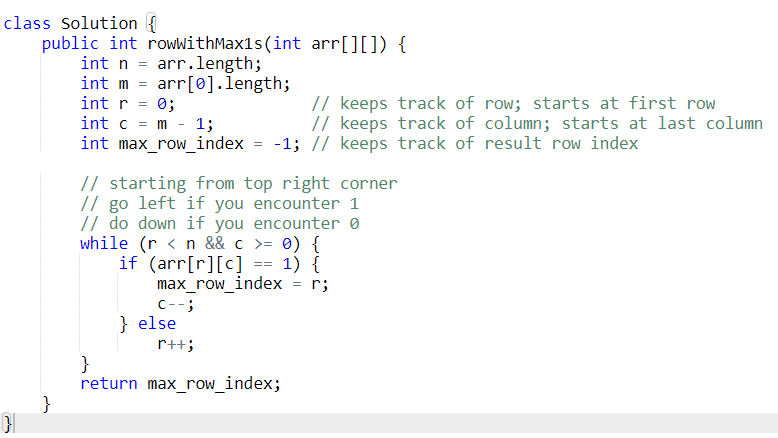
**Output: 2**

**Explanation: Row 2 contains 4 1's.**

**The intuition is to find the row in a 2D matrix (represented as a vector of vectors) with the maximum number of 1s. It iterates through each row and counts the number of 1s in each row. The row with the highest count of 1s is identified, and its index is returned as the output.**

****

**The problem involves searching for the row with the maximum number of 1s in a boolean 2D array where each row is sorted. Since the rows are sorted, we can utilize this property to find the desired row efficiently. The algorithm starts from the top-right corner of the array and traverses the array while moving left if the current element is 1 or moving down if the current element is 0. This way, the algorithm narrows down the rows that could potentially have the maximum number of 1s. Eventually, it stops at the row with the maximum number of 1s.**

****

**Q. 19 Maximum Product Subarray**

[**https://www.geeksforgeeks.org/problems/maximum-product-subarray3604/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/maximum-product-subarray3604/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**Given an array arr[] that contains positive and negative integers (may contain 0 as well). Find the maximum product that we can get in a subarray of arr.**

**Note: It is guaranteed that the output fits in a 64-bit integer.**

**Examples**

**Input: arr[] = [6, -3, -10, 0, 2]**

**Output: 180**

**Explanation: The subarray [6, -3, -10] gives max product as 180.**

**Input: arr[] = [2, 3, 4, 5, -1, 0]**

**Output: 120**

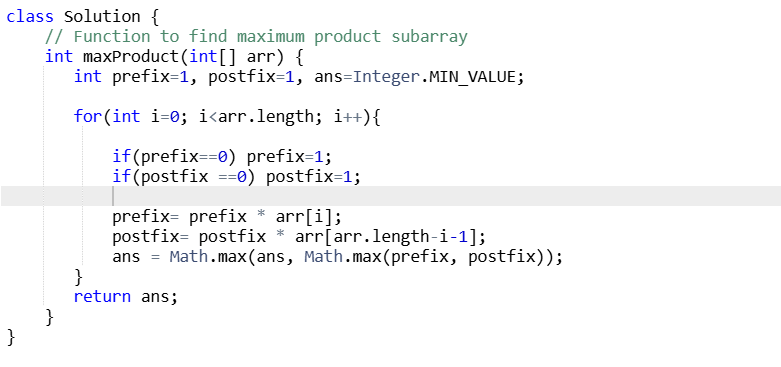
**Explanation: The subarray [2, 3, 4, 5] gives max product as 120.**

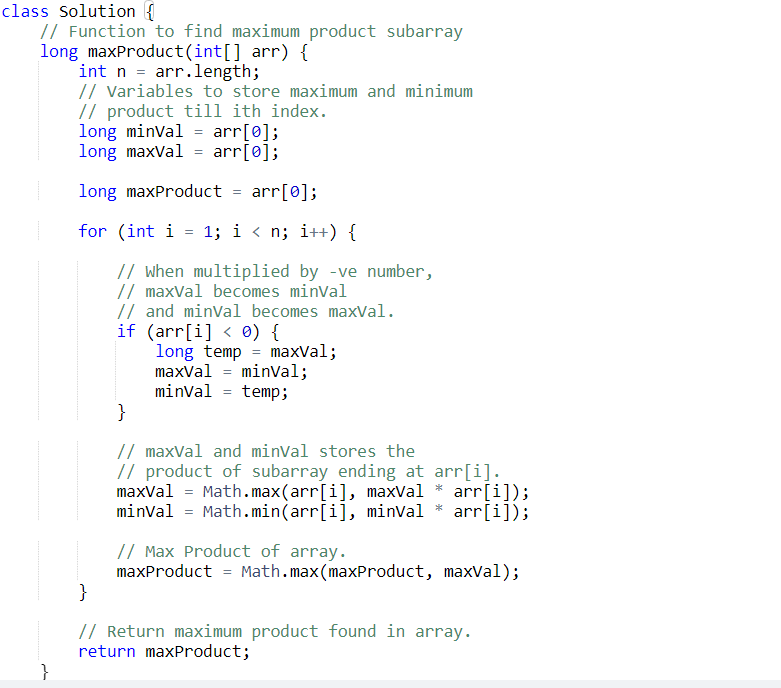
**Intuition**

**The intuition is to maintains two variable, maxVal and minVal, to track the maximum and minimum product of subarrays ending at the current index. It traverses through the array and updates these variables based on the current element's value. The goal is to handle negative numbers by swapping maxVal and minVal when encountering a negative element to ensure that the maximum product is accurately calculated.**

**Implementation**

1. **Initialize minVal, maxVal, and maxProduct with the value of the first element of the array.**
2. **Start iterating through the array from the second element (index 1) up to the last element.**
3. **Inside the loop:**
   * **If the current element is negative, swap maxVal and minVal. This swap handles negative numbers to ensure that maxVal holds the maximum product and minVal holds the minimum product of subarrays ending at the current index.**
   * **Update maxVal and minVal by taking the maximum and minimum between the current element and the product of the current element with the respective values (maxVal and minVal) from the previous index.**
   * **Update maxProduct by taking the maximum between maxProduct and maxVal.**
4. **After the loop completes, return maxProduct, which holds the maximum product of a subarray within the given array.**

****

****

**Q.20 Minimize the Heights I**

[**https://www.geeksforgeeks.org/problems/minimize-the-heights-i/1**](https://www.geeksforgeeks.org/problems/minimize-the-heights-i/1)

**Given a positive integer k and an array arr[] denoting heights of towers, you have to modify the height of each tower either by increasing or decreasing them by k only once.  
Find out what could be the possible minimum difference of the height of shortest and longest towers after you have modified each tower.**

**Note: A slight modification of the problem can be found**[**here**](https://practice.geeksforgeeks.org/problems/minimize-the-heights3351/1)**.   
  
Examples:**

**Input: k = 2, arr[] = [1, 5, 8, 10]**

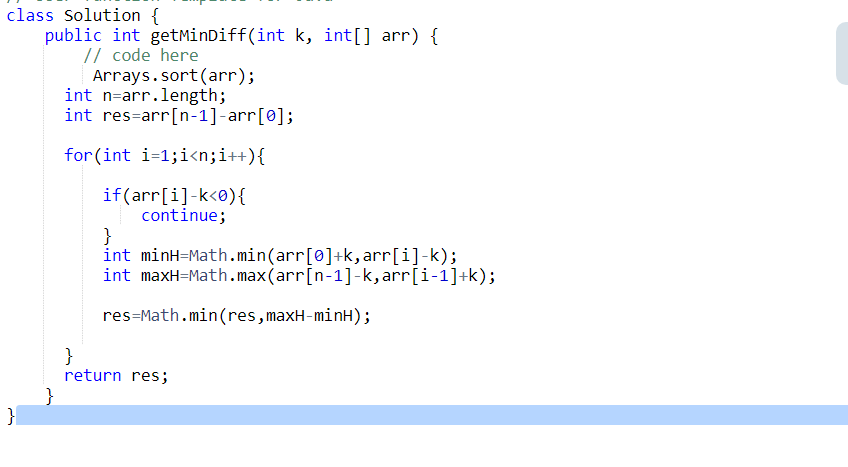
**Output: 5**

**Explanation: The array can be modified as [3, 3, 6, 8]. The difference between the largest and the smallest is 8 - 3 = 5.**

**Input: k = 3, arr[] = [3, 9, 12, 16, 20]**

**Output: 11**

**Explanation: The array can be modified as [6, 12, 9, 13, 17]. The difference between the largest and the smallest is 17 - 6 = 11.**

****

**Q.21 Alternative Sorting**

[**https://www.geeksforgeeks.org/problems/alternative-sorting1311/1**](https://www.geeksforgeeks.org/problems/alternative-sorting1311/1)

**Given an array arr of distinct integers. Rearrange the array in such a way that the first element is the largest and the second element is the smallest, the third element is the second largest and the fourth element is the second smallest, and so on.**

**Examples:**

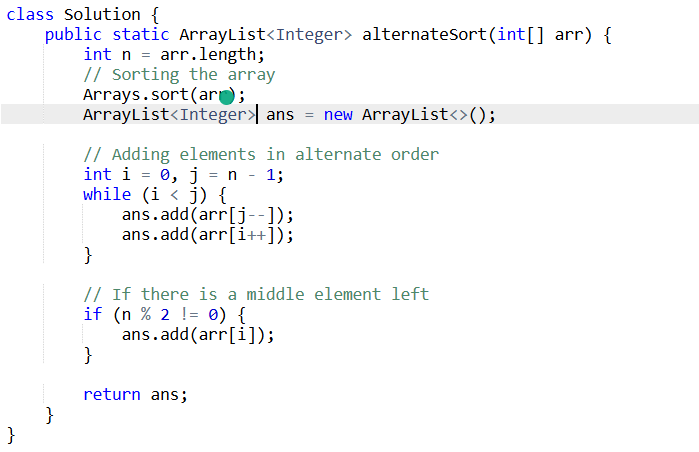
**Input: arr[] = [7, 1, 2, 3, 4, 5, 6]**

**Output: [7, 1, 6, 2, 5, 3, 4]**

**Explanation: The first element is first maximum and second element is first minimum and so on.**

**Input: arr[] = [1, 6, 9, 4, 3, 7, 8, 2]**

**Output: [9, 1, 8, 2, 7, 3, 6, 4]  
Explanation: The first element is first maximum and second element is first minimum and so on.**

****

**Q.22 Minimize the Heights II**

[**https://www.geeksforgeeks.org/problems/minimize-the-heights3351/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty**](https://www.geeksforgeeks.org/problems/minimize-the-heights3351/1?page=1&category=Arrays&sprint=50746f92a895c22a50504ac0c1fb9c84&sortBy=difficulty)

**Given an array arr[] denoting heights of N towers and a positive integer K.**

**For each tower, you must perform exactly one of the following operations exactly once.**

* **Increase the height of the tower by K**
* **Decrease the height of the tower by K**

**Find out the minimum possible difference between the height of the shortest and tallest towers after you have modified each tower.**

**You can find a slight modification of the problem**[**here**](https://practice.geeksforgeeks.org/problems/minimize-the-heights-i/1/)**.  
Note: It is compulsory to increase or decrease the height by K for each tower. After the operation, the resultant array should not contain any negative integers.**

**Examples :**

**Input: k = 2, arr[] = {1, 5, 8, 10}**

**Output: 5**

**Explanation: The array can be modified as {1+k, 5-k, 8-k, 10-k} = {3, 3, 6, 8}.The difference between the largest and the smallest is 8-3 = 5.**

**Input: k = 3, arr[] = {3, 9, 12, 16, 20}**

**Output: 11**

**Explanation: The array can be modified as {3+k, 9+k, 12-k, 16-k, 20-k} -> {6, 12, 9, 13, 17}.The difference between the largest and the smallest is 17-6 = 11.**

**The idea for this is given below:**

* **The idea is to increase the first i towers by k and decrease the rest tower by k after sorting the heights, then calculate the maximum height difference.**
* **This can be achieved using sorting.**

**Illustration:**

**Given arr[] = {1, 15, 10}, n = 3, k = 6**

**Array after sorting => arr[] = {1, 10, 15}**

**Initially maxHeight = arr[n – 1] = 15  
            minHeight = arr[0] = 1  
            ans = maxHeight – minHeight = 15 – 1 = 14**

**At i = 1**

* **minHeight = min(arr[0] + k, arr[i] – k) = min(1 + 6, 10 – 6) = 4**
* **maxHeight = max(arr[i – 1] + k, arr[n – 1] – k) = max(1 + 6, 15 – 6) = 9**
* **ans = min(ans, maxHeight – minHeight) = min(14, 9 – 4) = 5 => ans = 5**

**At i = 2**

* **minHeight = min(arr[0] + k, arr[i] – k) = min(1 + 6, 15 – 6) = 7**
* **maxHeight = max(arr[i – 1] + k, arr[n – 1] – k) = max(10 + 6, 15 – 6) = 16**
* **ans = min(ans, maxHeight – minHeight) = min(5, 16 – 7) = 5 => ans = 5**

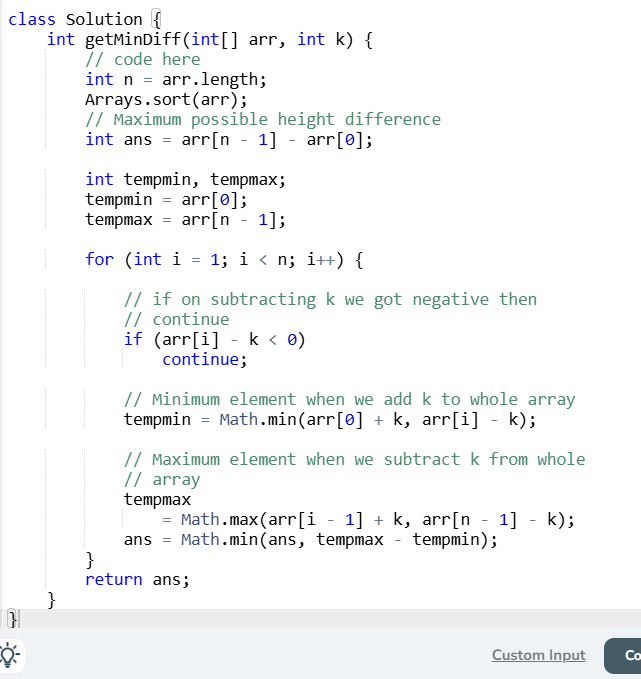
**Hence minimum difference is 5**

**Note:- Consider where a[i] < K because the height of the tower can’t be negative so neglect that case. You may wonder that if we neglect this case, then we would also be neglecting a[i-1] + k; what if it is greater than a[n-1]-k? The answer for that is because a[i] < K, we don’t have any other option than to increase its height by K. And because a[i] > a[i-1], hence a[i] + k would also be greater than a[i-1]+k. Therefore, a[i-1] + k would never be the maximum height of the array and hence can be neglected.**

**Furthermore, the reason we don’t take a[i] for both minHeight and maxHeight is because it is possible that a[i] – k < arr[0] +k and at the same time a[i] +k > a[n-1] – k. In this scenario, we would be both increasing and decreasing the height of the tower which is not possible.**

**Follow the steps below to solve the given problem:**

* **Sort the array**
* **Try to make each height of the tower maximum by decreasing the height of all the towers to the right by k and increasing all the height of the towers to the left by k. Check whether the current index tower has the maximum height or not by comparing it with a[n]-k. If the tower’s height is greater than the a[n]-k then it’s the tallest tower available.**
* **Similarly, find the shortest tower and minimize the difference between these two towers.**

****