1.Coin Change (Count Ways)

Given an integer array coins[ ] representing different denominations of currency and an integer sum, find the number of ways you can make sum by using different combinations from coins[ ].
Note: Assume that you have an infinite supply of each type of coin. And you can use any coin as many times as you want.
Answers are guaranteed to fit into a 32-bit integer.

Examples:

Input: coins[] = [1, 2, 3], sum = 4

Output: 4

Explanation: Four Possible ways are: [1, 1, 1, 1], [1, 1, 2], [2, 2], [1, 3].

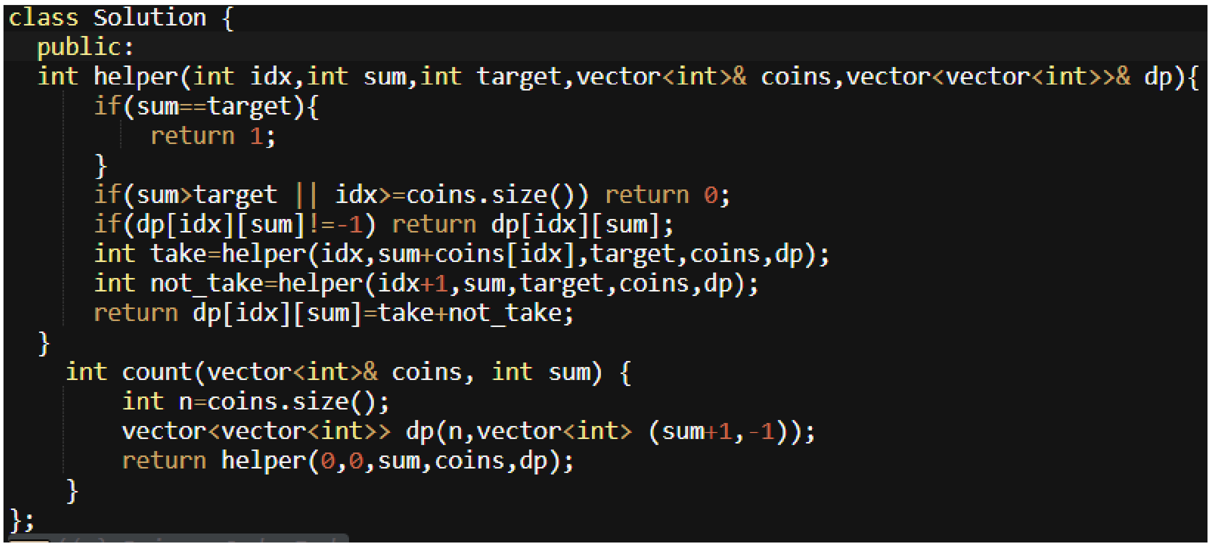
Input: coins[] = [2, 5, 3, 6], sum = 10

Output: 5

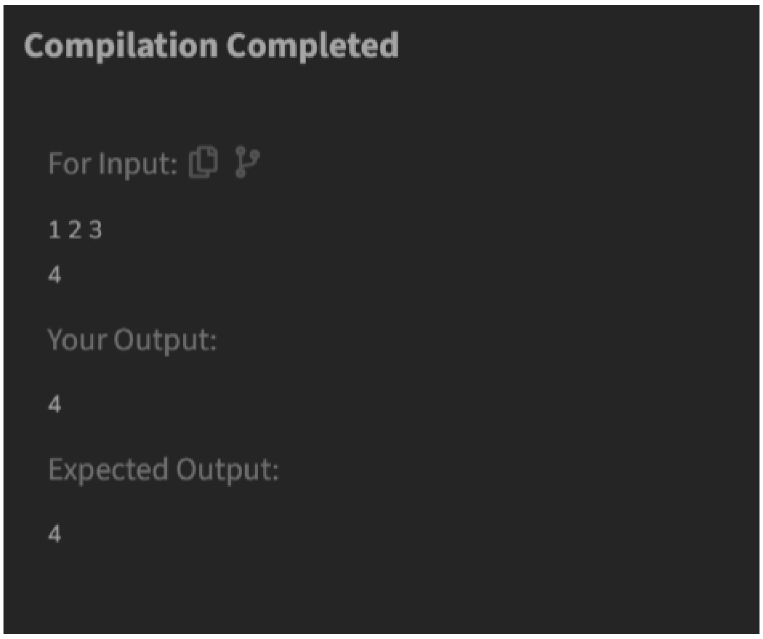
Explanation: Five Possible ways are: [2, 2, 2, 2, 2], [2, 2, 3, 3], [2, 2, 6], [2, 3, 5] and [5, 5].

Input: coins[] = [5, 10], sum = 3

Output: 0
Explanation: Since all coin denominations are greater than sum, no combination can make the target sum.



```
class Solution {
  public:
  int helper(int idx,int sum,int target,vector<int>& coins,vector<vector<int>>& dp){
      if(sum==target){
          return 1;
      }
      if(sum>target || idx>=coins.size()) return 0;
      if(dp[idx][sum]!=-1) return dp[idx][sum];
      int take=helper(idx,sum+coins[idx],target,coins,dp);
      int not_take=helper(idx+1,sum,target,coins,dp);
      return dp[idx][sum]=take+not_take;
  }
    int count(vector<int>& coins, int sum) {
        int n=coins.size();
        vector<vector<int>> dp(n,vector<int> (sum+1,-1));
        return helper(0,0,sum,coins,dp);
    }
};
```


```
Compilation Completed

For Input:

1 2 3
4

Your Output:

4

Expected Output:

4
```

Time Complexity: O(sum*n)

Auxiliary Space: O(sum*n)

2.Stock buy and sell

The cost of stock on each day is given in an array **A**[] of size **N**. Find all the segments of days on which you buy and sell the stock such that the sum of difference between sell and buy prices is maximized. Each segment consists of indexes of two elements, first is index of day on which you buy stock and second is index of day on which you sell stock.
Note: Since there can be multiple solutions, the driver code will print 1 if your answer is correct, otherwise, it will return 0. In case there's no profit the driver code will print the string "**No Profit**" for a correct solution.

Example 1:

Input: N = 7 A[] = {100,180,260,310,40,535,695}

Output:1

Explanation:

One possible solution is (0 3) (4 6) We can buy stock on day 0,and sell it on 3rd day, which will give us maximum profit. Now, we buy stock on day 4 and sell it on day 6.

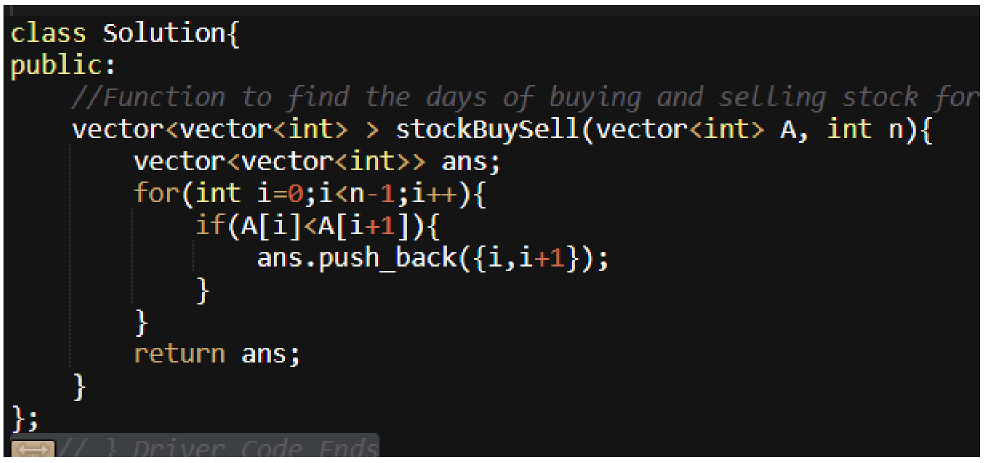
Example 2:

Input: N = 5 A[] = {4,2,2,2,4}

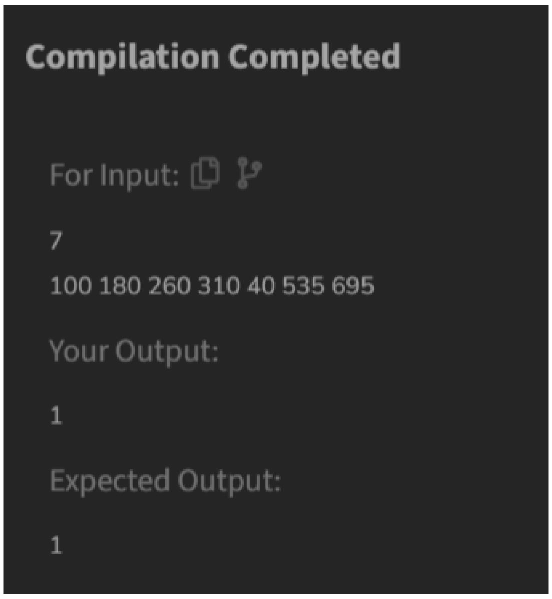
Output:1

Explanation:

There are multiple possible solutions.one of them is (3 4)We can buy stock on day 3,and sell it on 4th day, which will give us maximum profit.



```
class Solution{
public:
    //Function to find the days of buying and selling stock for
    vector<vector<int> > stockBuySell(vector<int> A, int n){
        vector<vector<int>> ans;
        for(int i=0;i<n-1;i++){
            if(A[i]<A[i+1]){
                ans.push_back({i,i+1});
            }
        }
        return ans;
    }
};
// } Driver Code Ends
```



```
7
100 180 260 310 40 535 695
```

```
1
```

```
1
```

Time Complexity: O(n)

Auxiliary Space: O(n)

3.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]

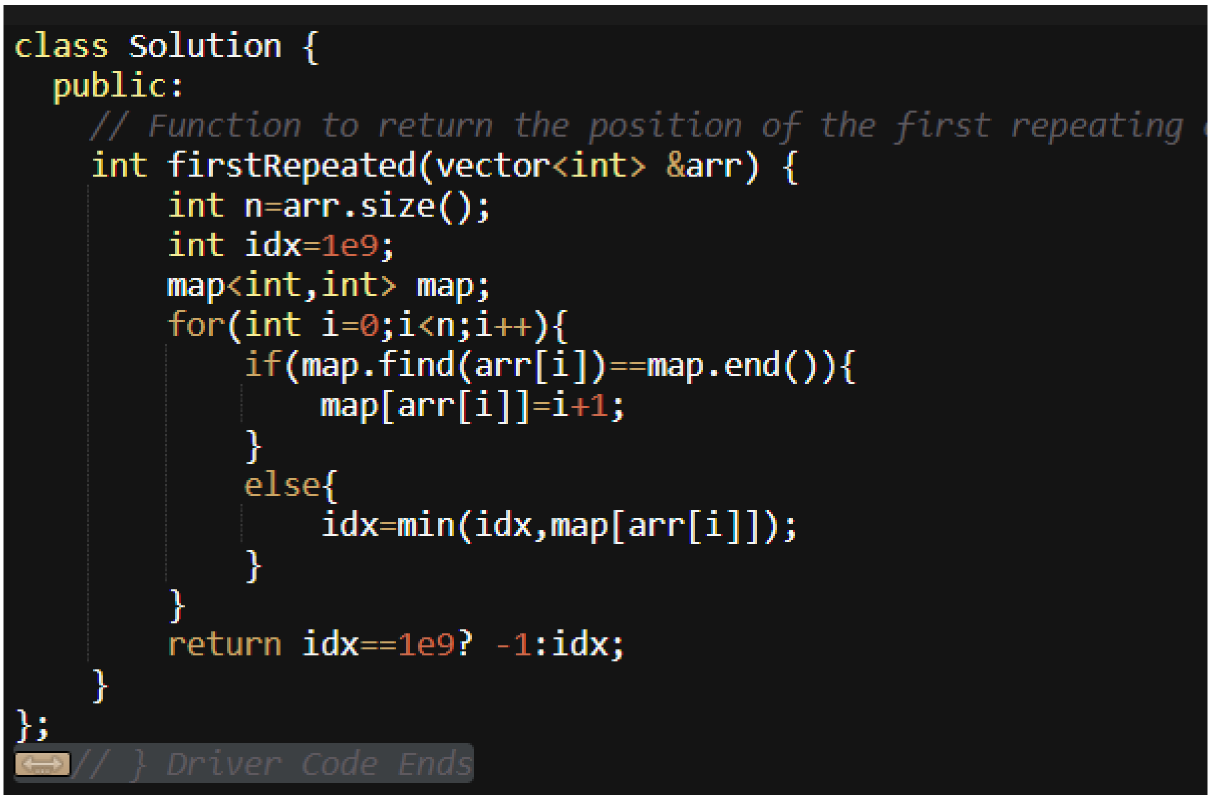
Output: 2

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

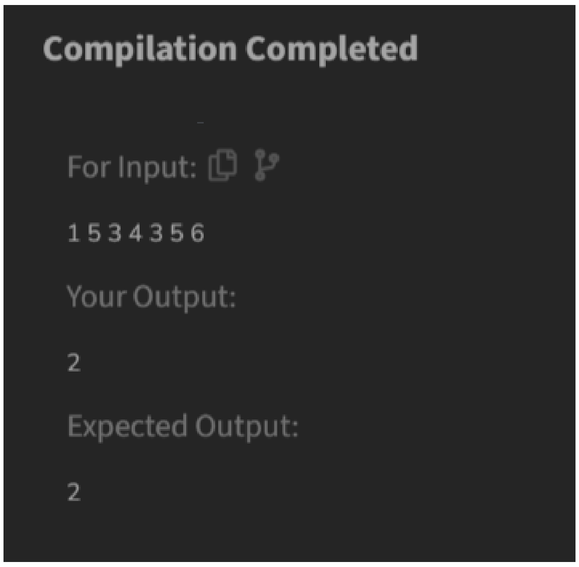
Input: arr[] = [1, 2, 3, 4]

Output: -1

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



```cpp
class Solution {
  public:
    // Function to return the position of the first repeating
    int firstRepeated(vector<int> &arr) {
        int n=arr.size();
        int idx=1e9;
        map<int,int> map;
        for(int i=0;i<n;i++){
            if(map.find(arr[i])==map.end()){
                map[arr[i]]=i+1;
            }
            else{
                idx=min(idx,map[arr[i]]);
            }
        }
        return idx==1e9? -1:idx;
    }
};
// } Driver Code Ends
```



Compilation Completed

```
For Input:
1 5 3 4 3 5 6
Your Output:
2
Expected Output:
2
```

Time Complexity: O(n)

Auxiliary Space: O(n)

4.First and Last Occurrences

Given a sorted array **arr** with possibly some duplicates, the task is to find the first and last occurrences of an element **x** in the given array.
Note: If the number **x** is not found in the array then return both the indices as -1.

Examples:

Input: arr[] = [1, 3, 5, 5, 5, 5, 67, 123, 125], x = 5

Output: [2, 5]

Explanation: First occurrence of 5 is at index 2 and last occurrence of 5 is at index 5

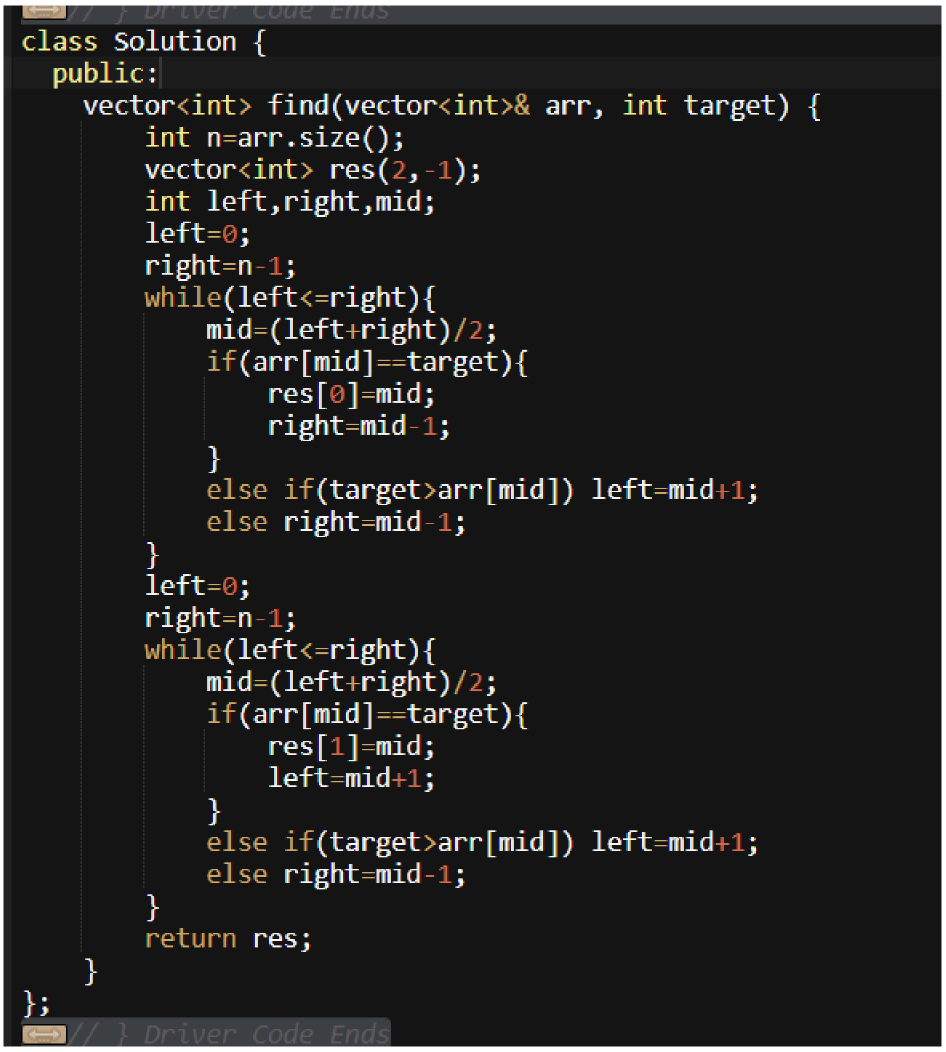
Input: arr[] = [1, 3, 5, 5, 5, 5, 7, 123, 125], x = 7

Output: [6, 6]
Explanation: First and last occurrence of 7 is at index 6

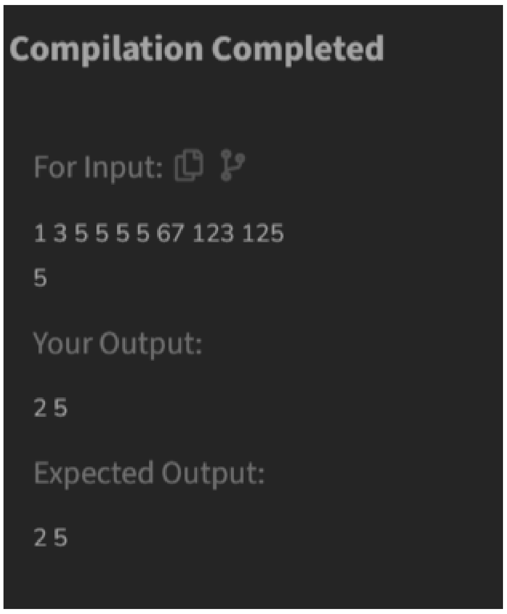
Input: arr[] = [1, 2, 3], x = 4

Output: [-1, -1]

Explanation: No occurrence of 4 in the array, so, output is [-1, -1]



```
// } Driver Code Ends
class Solution {
  public:
    vector<int> find(vector<int>& arr, int target) {
        int n=arr.size();
        vector<int> res(2,-1);
        int left,right,mid;
        left=0;
        right=n-1;
        while(left<=right){
            mid=(left+right)/2;
            if(arr[mid]==target){
                res[0]=mid;
                right=mid-1;
            }
            else if(target>arr[mid]) left=mid+1;
            else right=mid-1;
        }
        left=0;
        right=n-1;
        while(left<=right){
            mid=(left+right)/2;
            if(arr[mid]==target){
                res[1]=mid;
                left=mid+1;
            }
            else if(target>arr[mid]) left=mid+1;
            else right=mid-1;
        }
        return res;
    }
};
// } Driver Code Ends
```



```
1 3 5 5 5 5 67 123 125
5
```

```
2 5
```

```
2 5
```

Time Complexity: O(log n)

Auxiliary Space: O(1)

5.Remove Duplicates Sorted Array

Given a **sorted** array **arr.** Return the size of the modified array which contains only distinct elements.
Note:
1. Don't use set or HashMap to solve the problem.
2. You **must** return the modified array **size only** where distinct elements are present and **modify** the original array such that all the distinct elements come at the beginning of the original array.

Examples :

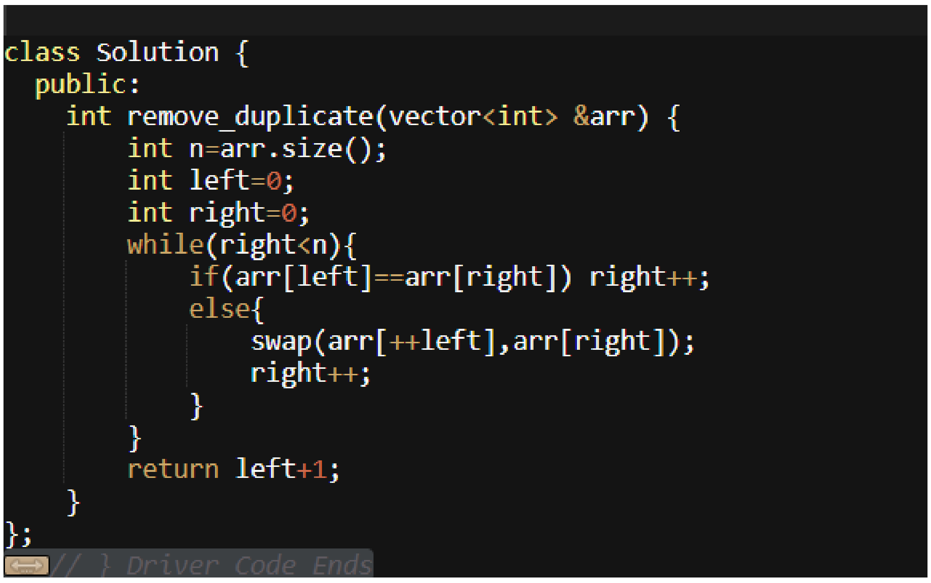
Input: arr = [2, 2, 2, 2, 2]

Output: [2]

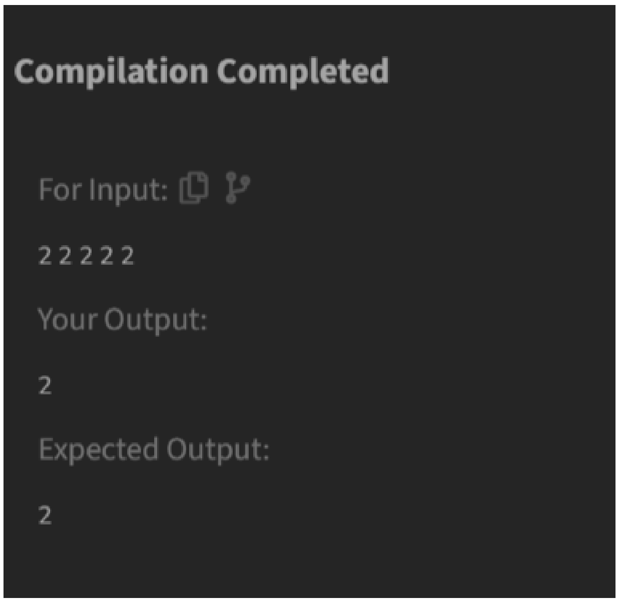
Explanation: After removing all the duplicates only one instance of 2 will remain i.e. [2] so modified array will contains 2 at first position and you should **return 1** after modifying the array, the driver code will print the modified array elements.

Input: arr = [1, 2, 4]

Output: [1, 2, 4]
Explation: As the array does not contain any duplicates so you should return 3.



```cpp
class Solution {
  public:
    int remove_duplicate(vector<int> &arr) {
        int n=arr.size();
        int left=0;
        int right=0;
        while(right<n){
            if(arr[left]==arr[right]) right++;
            else{
                swap(arr[++left],arr[right]);
                right++;
            }
        }
        return left+1;
    }
};
// } Driver Code Ends
```



```
Compilation Completed

For Input:
2 2 2 2 2
Your Output:
2
Expected Output:
2
```

Time Complexity: O(n)

Auxiliary Space: O(1)

6.Maximum Index

Given an array arr of positive integers. The task is to return the maximum of j - i subjected to the constraint of arr[i] ≤ arr[j] and i ≤ j.

Examples:

Input: arr[] = [1, 10]

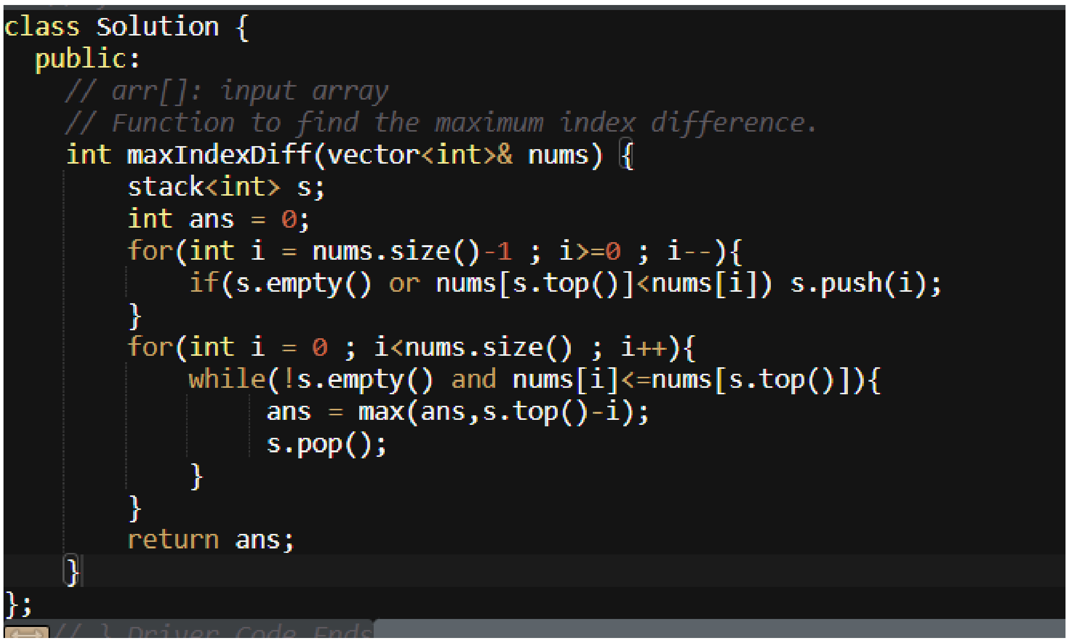
Output: 1

Explanation: arr[0] ≤ arr[1] so (j-i) is 1-0 = 1.

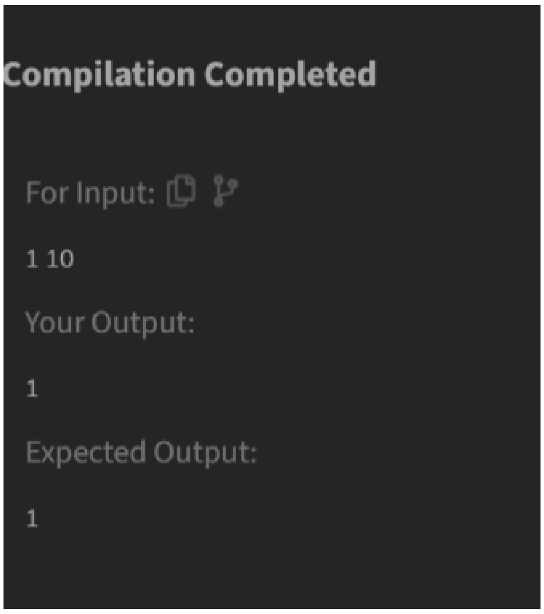
Input: arr[] = [34, 8, 10, 3, 2, 80, 30, 33, 1]

Output: 6

Explanation: In the given array arr[1] < arr[7] satisfying the required condition(arr[i] ≤ arr[j]) thus giving the maximum difference of j - i which is 6(7-1).



```
class Solution {
  public:
    // arr[]: input array
    // Function to find the maximum index difference.
    int maxIndexDiff(vector<int>& nums) {
        stack<int> s;
        int ans = 0;
        for(int i = nums.size()-1 ; i>=0 ; i--){
            if(s.empty() or nums[s.top()]<nums[i]) s.push(i);
        }
        for(int i = 0 ; i<nums.size() ; i++){
            while(!s.empty() and nums[i]<=nums[s.top()]){
                ans = max(ans,s.top()-i);
                s.pop();
            }
        }
        return ans;
    }
};
// } Driver Code Ends
```



```
Compilation Completed

For Input:
1 10
Your Output:
1
Expected Output:
1
```

Time Complexity: O(n)

Auxiliary Space: O(n)

7.Wave Array

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

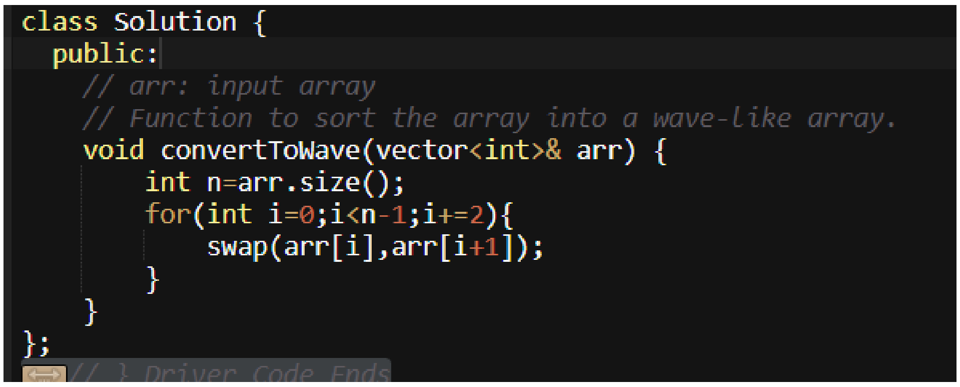
Output: [2, 1, 4, 3, 5]

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

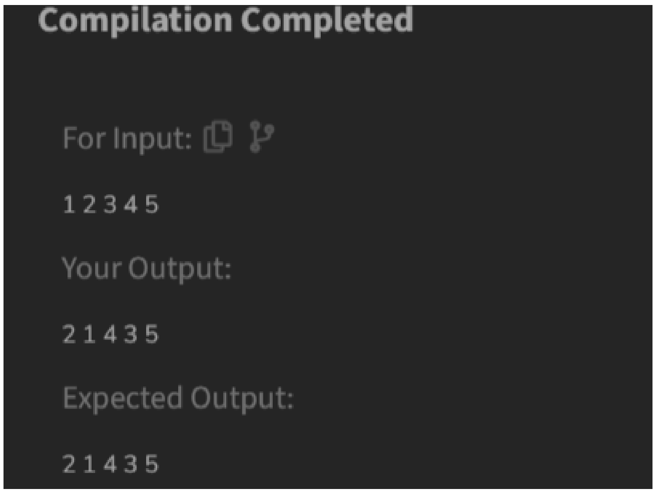
Input: arr[] = [2, 4, 7, 8, 9, 10]

Output: [4, 2, 8, 7, 10, 9]

Explanation: Array elements after sorting it in the waveform are 4, 2, 8, 7, 10, 9.
Input: arr[] = [1]
Output: [1]



```
class Solution {
  public:
    // arr: input array
    // Function to sort the array into a wave-like array.
    void convertToWave(vector<int>& arr) {
        int n=arr.size();
        for(int i=0;i<n-1;i+=2){
            swap(arr[i],arr[i+1]);
        }
    }
};
// } Driver Code Ends
```



```
Compilation Completed

For Input:

1 2 3 4 5

Your Output:

2 1 4 3 5

Expected Output:

2 1 4 3 5
```

Time Complexity: O(n)

Auxiliary Space: O(1)

8.Find Transition Point

Given a sorted array, arr[] containing only 0s and 1s, find the transition point, i.e., the first index where 1 was observed, and before that, only 0 was observed. If arr does not have any 1, return -1. If array does not have any 0, return 0.

Examples:

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

Output: 3

Explanation: index 3 is the transition point where 1 begins.

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

Output: -1

Explanation: Since, there is no "1", the answer is -1.

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

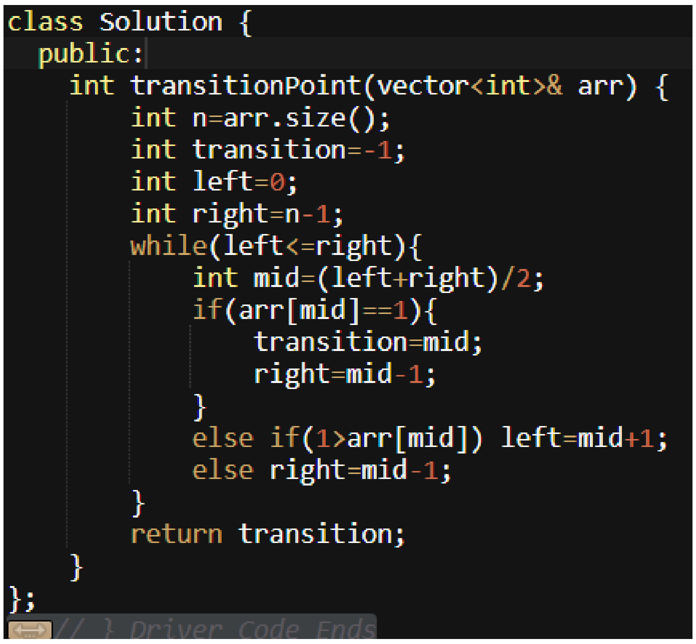
Output: 0

Explanation: There are no 0s in the array, so the transition point is 0, indicating that the first index (which contains 1) is also the first position of the array.

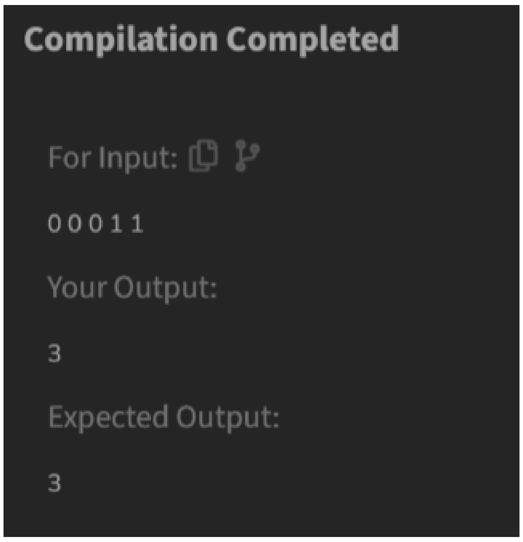
Input: arr[] = [0, 1, 1]

Output: 1

Explanation: Index 1 is the transition point where 1 starts, and before it, only 0 was observed.



```
class Solution {
  public:
    int transitionPoint(vector<int>& arr) {
        int n=arr.size();
        int transition=-1;
        int left=0;
        int right=n-1;
        while(left<=right){
            int mid=(left+right)/2;
            if(arr[mid]==1){
                transition=mid;
                right=mid-1;
            }
            else if(1>arr[mid]) left=mid+1;
            else right=mid-1;
        }
        return transition;
    }
};
// } Driver Code Ends
```



```
Compilation Completed

For Input:
0 0 0 1 1
Your Output:
3
Expected Output:
3
```

Time Complexity: O(log n)

Auxiliary Space: O(1)