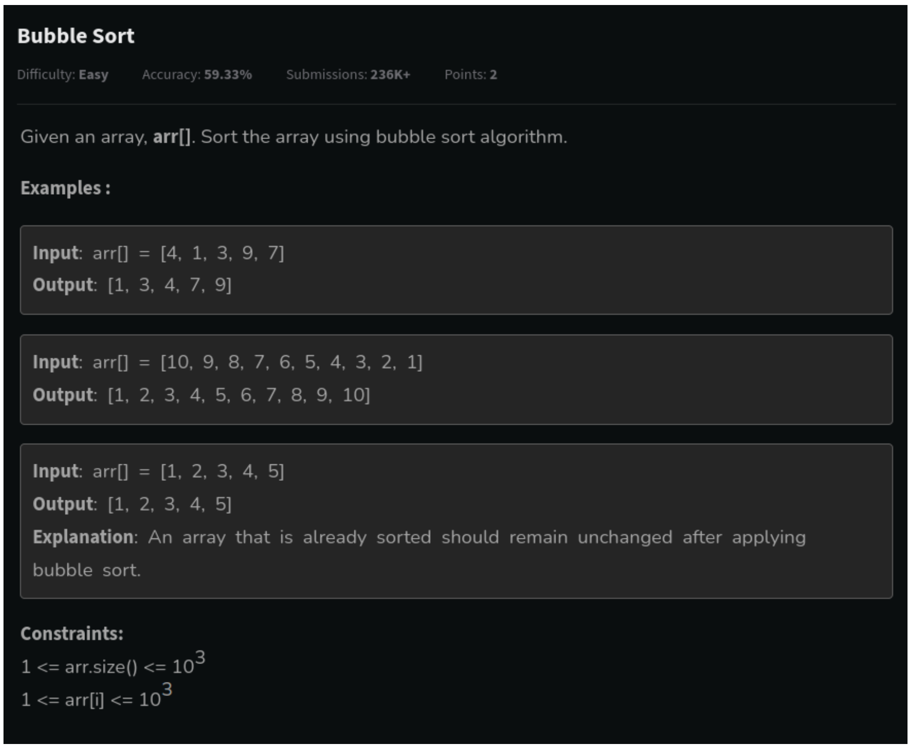


Bubble Sort

Difficulty: **Easy** Accuracy: **59.33%** Submissions: **236K+** Points: **2**

Given an array, **arr[]**. Sort the array using bubble sort algorithm.

Examples :

```
Input: arr[] = [4, 1, 3, 9, 7]
Output: [1, 3, 4, 7, 9]
```

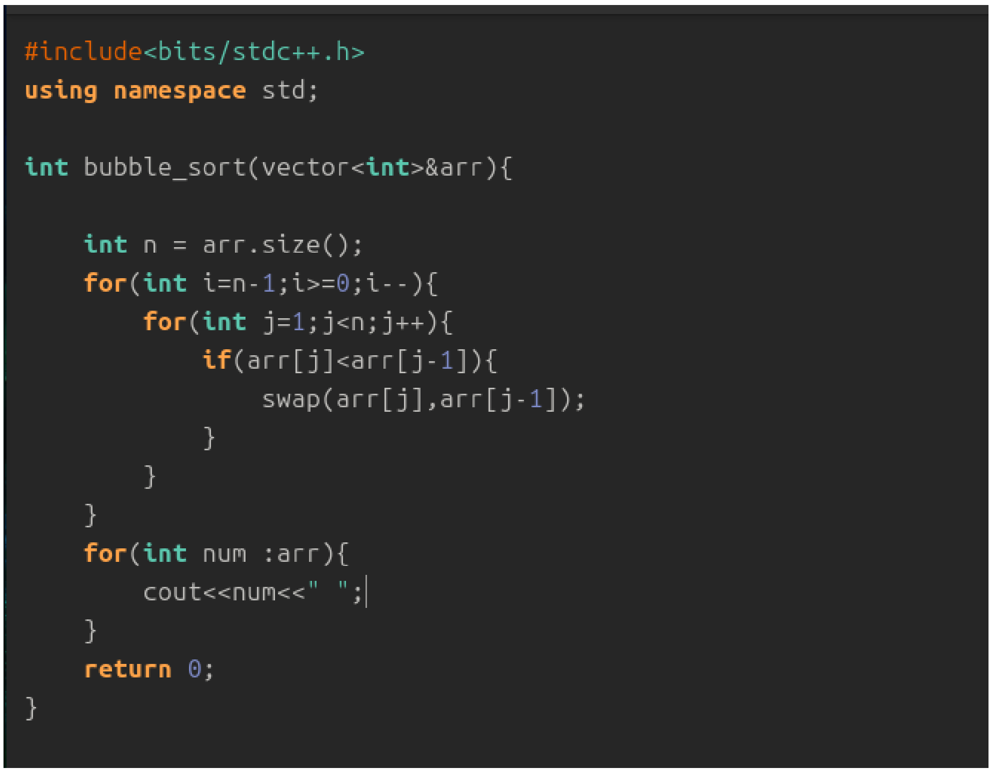
```
Input: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
Input: arr[] = [1, 2, 3, 4, 5]
Output: [1, 2, 3, 4, 5]
Explanation: An array that is already sorted should remain unchanged after applying
bubble sort.
```

Constraints:

1 <= arr.size() <= 10^3

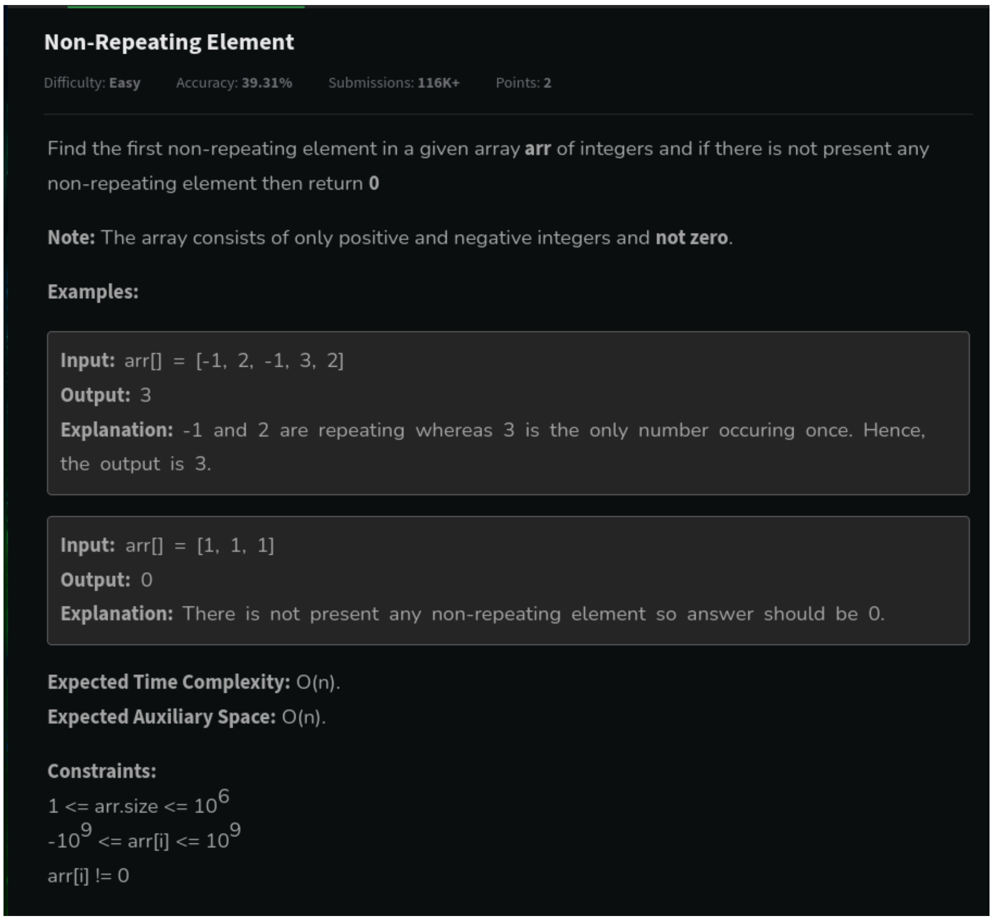
1 <= arr[i] <= 10^3



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

int bubble_sort(vector<int>&arr){

    int n = arr.size();
    for(int i=n-1;i>=0;i--){
        for(int j=1;j<n;j++){
            if(arr[j]<arr[j-1]){
                swap(arr[j],arr[j-1]);
            }
        }
    }
    for(int num :arr){
        cout<<num<<" ";
    }
    return 0;
}
```

TimeComplexity : O(n^2)

Space Complexity : O(1)


Non-Repeating Element

Difficulty: **Easy** Accuracy: **39.31%** Submissions: **116K+** Points: **2**

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**.

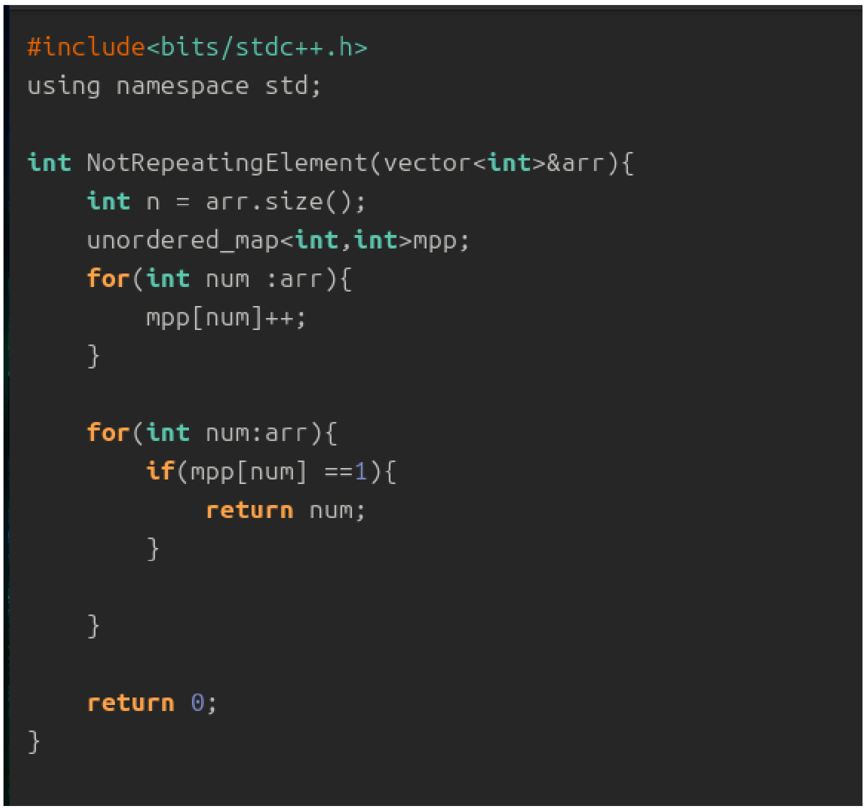
Examples:

```
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.
```

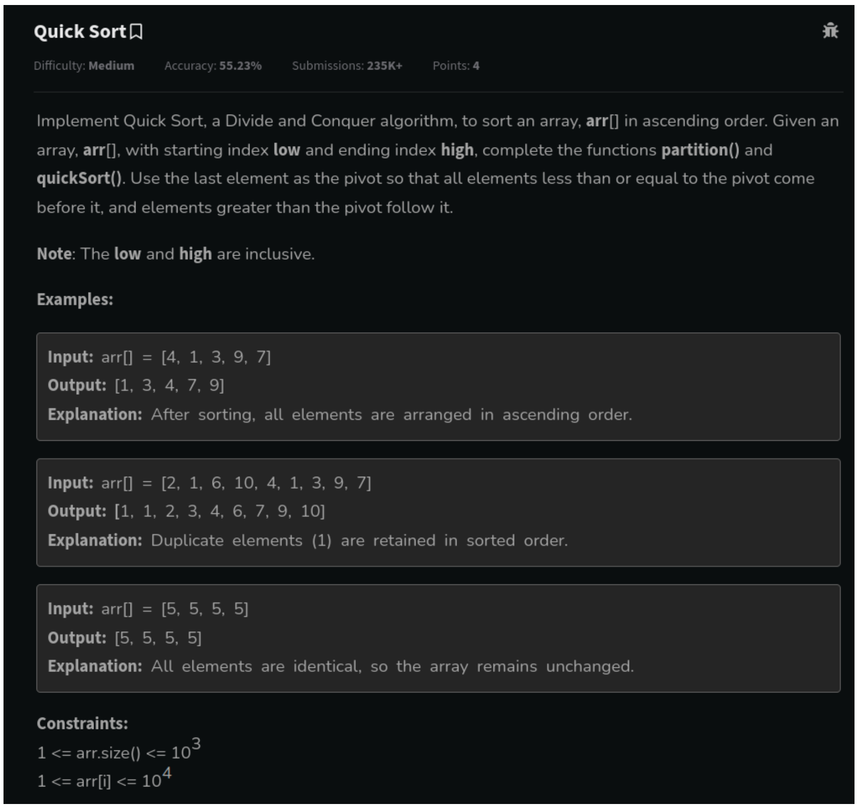
Expected Time Complexity: O(n).
Expected Auxiliary Space: O(n).

Constraints:
1 <= arr.size <= 10^6
-10^9 <= arr[i] <= 10^9
arr[i] != 0


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

int NotRepeatingElement(vector<int>&arr){
    int n = arr.size();
    unordered_map<int,int>mpp;
    for(int num :arr){
        mpp[num]++;
    }

    for(int num:arr){
        if(mpp[num] ==1){
            return num;
        }

    }

    return 0;
}
```

Time Complexity : O(n)

Space Complexity: O(n)



Quick Sort

Difficulty: **Medium** Accuracy: **55.23%** Submissions: **235K+** Points: **4**

Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, **arr[]** in ascending order. Given an array, **arr[]**, with starting index **low** and ending index **high**, complete the functions **partition()** and **quickSort()**. Use the last element as the pivot so that all elements less than or equal to the pivot come before it, and elements greater than the pivot follow it.

Note: The **low** and **high** are inclusive.

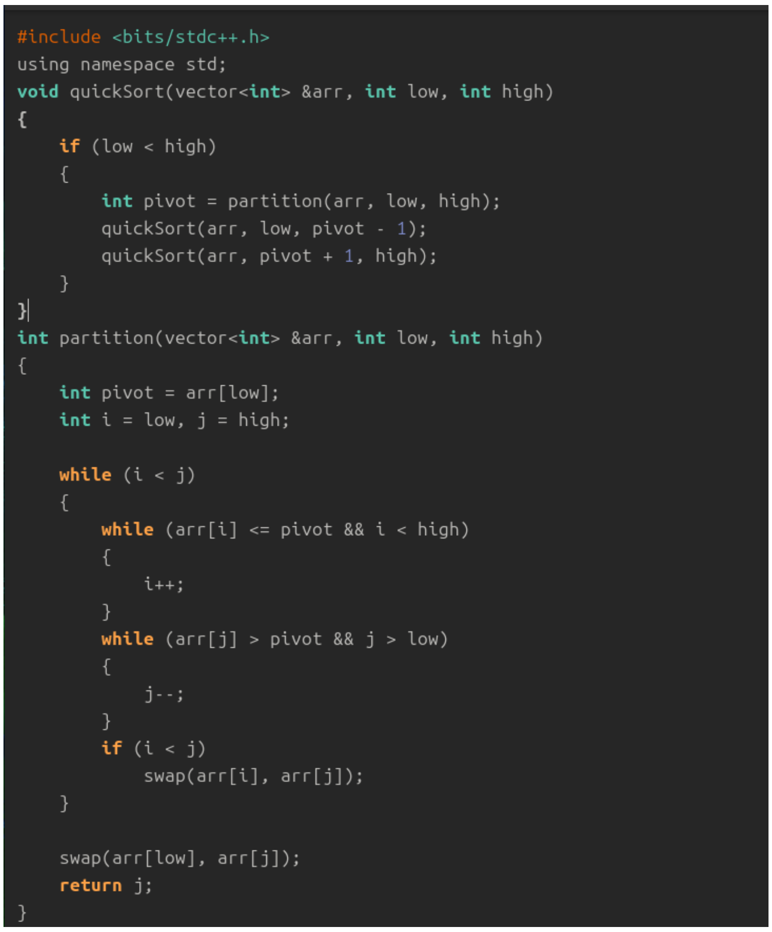
Examples:

```
Input: arr[] = [4, 1, 3, 9, 7]
Output: [1, 3, 4, 7, 9]
Explanation: After sorting, all elements are arranged in ascending order.
```

```
Input: arr[] = [2, 1, 6, 10, 4, 1, 3, 9, 7]
Output: [1, 1, 2, 3, 4, 6, 7, 9, 10]
Explanation: Duplicate elements (1) are retained in sorted order.
```

```
Input: arr[] = [5, 5, 5, 5]
Output: [5, 5, 5, 5]
Explanation: All elements are identical, so the array remains unchanged.
```

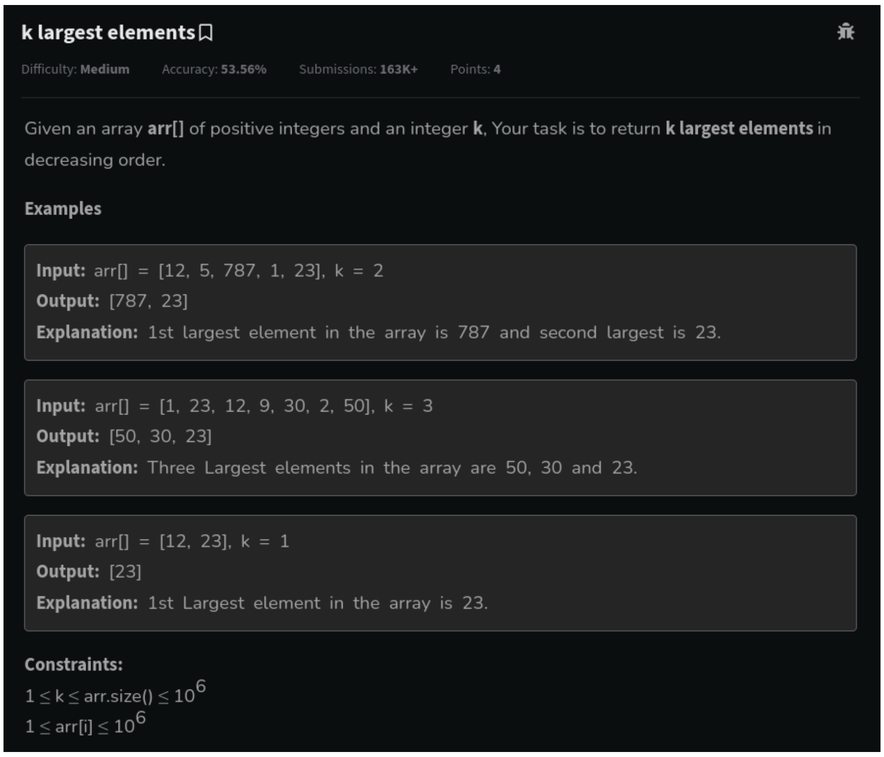
Constraints:
1 <= arr.size() <= 10^3
1 <= arr[i] <= 10^4



```cpp
#include <bits/stdc++.h>
using namespace std;
void quickSort(vector<int> &arr, int low, int high)
{
    if (low < high)
    {
        int pivot = partition(arr, low, high);
        quickSort(arr, low, pivot - 1);
        quickSort(arr, pivot + 1, high);
    }
}
int partition(vector<int> &arr, int low, int high)
{
    int pivot = arr[low];
    int i = low, j = high;

    while (i < j)
    {
        while (arr[i] <= pivot && i < high)
        {
            i++;
        }
        while (arr[j] > pivot && j > low)
        {
            j--;
        }
        if (i < j)
            swap(arr[i], arr[j]);
    }

    swap(arr[low], arr[j]);
    return j;
}
```

Time Complexity : O(N Log N)

Space Complexity : O(1)



k largest elements

Difficulty: **Medium** Accuracy: **53.56%** Submissions: **163K+** Points: **4**

Given an array **arr[]** of positive integers and an integer **k**, Your task is to return **k largest elements** in decreasing order.

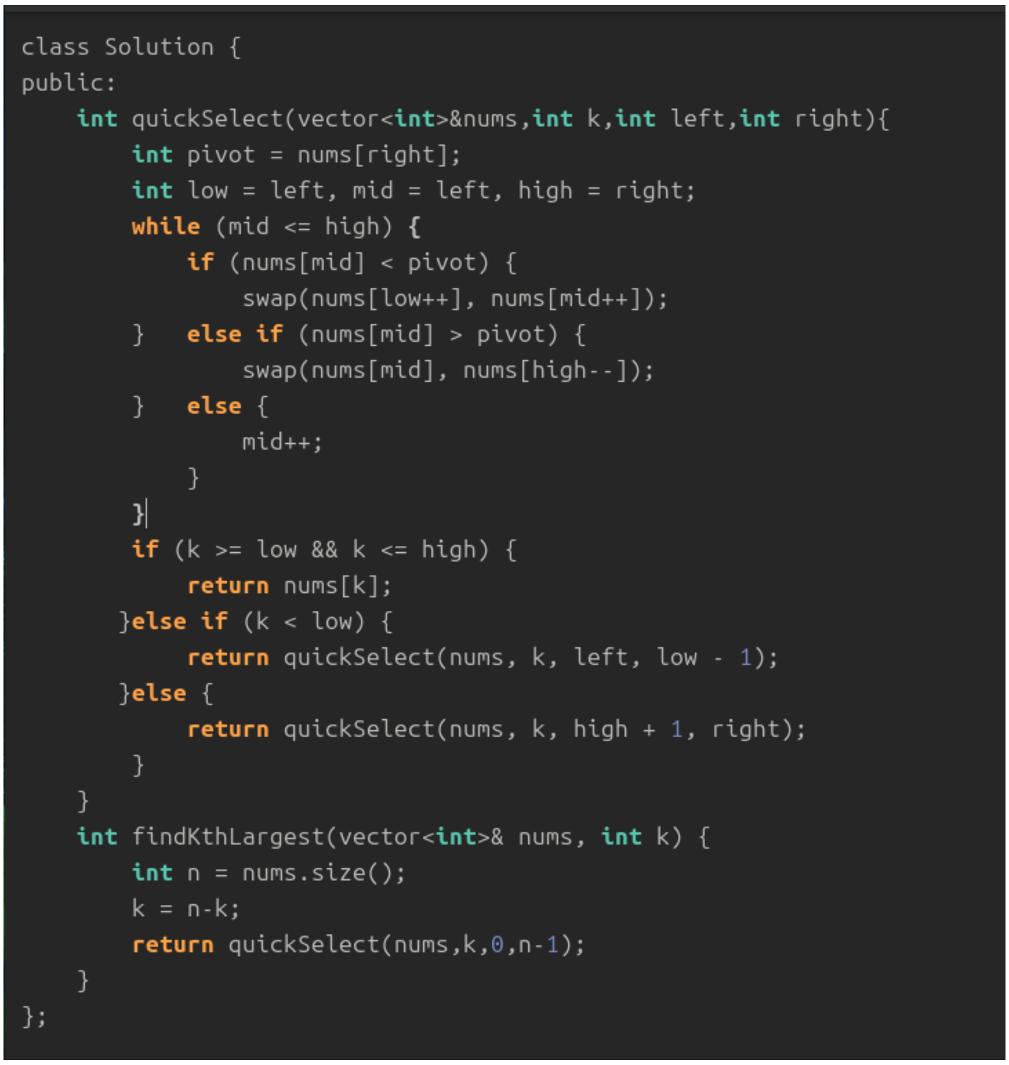
Examples

Input: arr[] = [12, 5, 787, 1, 23], k = 2
Output: [787, 23]
Explanation: 1st largest element in the array is 787 and second largest is 23.

Input: arr[] = [1, 23, 12, 9, 30, 2, 50], k = 3
Output: [50, 30, 23]
Explanation: Three Largest elements in the array are 50, 30 and 23.

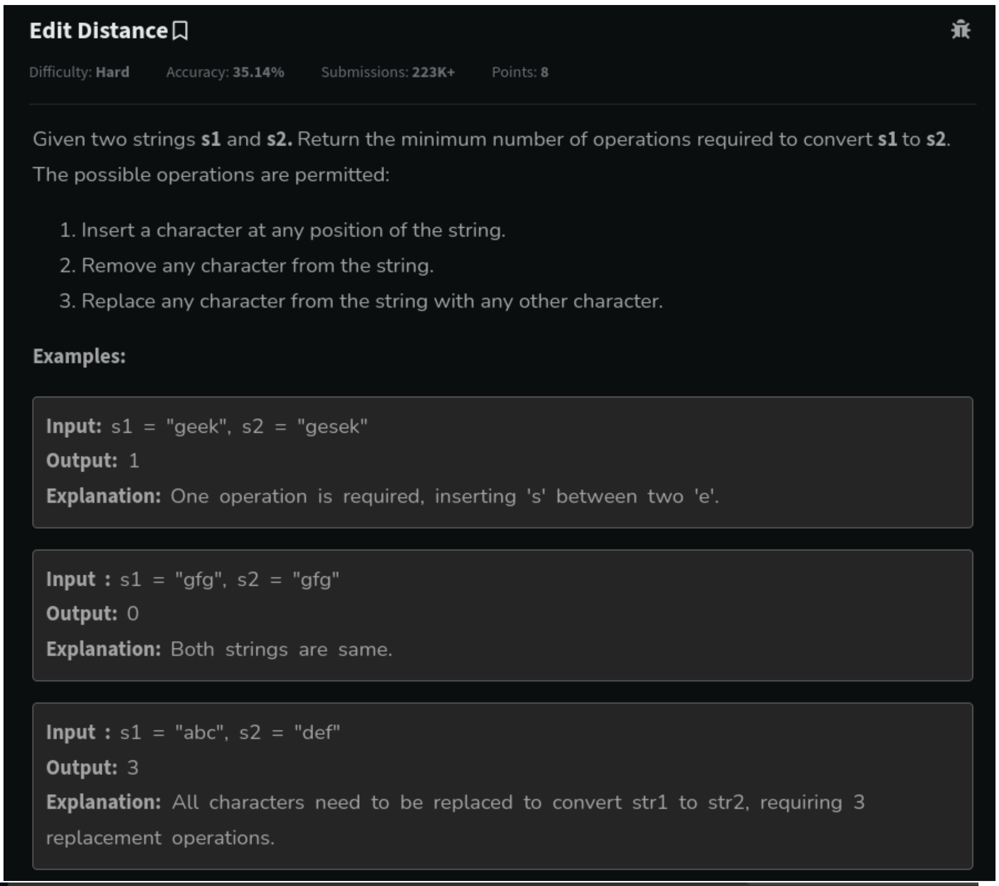
Input: arr[] = [12, 23], k = 1
Output: [23]
Explanation: 1st Largest element in the array is 23.

Constraints:
$1 \le k \le arr.size() \le 10^6$
$1 \le arr[i] \le 10^6$



```cpp
class Solution {
public:
    int quickSelect(vector<int>&nums,int k,int left,int right){
        int pivot = nums[right];
        int low = left, mid = left, high = right;
        while (mid <= high) {
            if (nums[mid] < pivot) {
                swap(nums[low++], nums[mid++]);
        }   else if (nums[mid] > pivot) {
                swap(nums[mid], nums[high--]);
        }   else {
                mid++;
            }
        }
        if (k >= low && k <= high) {
            return nums[k];
       }else if (k < low) {
            return quickSelect(nums, k, left, low - 1);
       }else {
            return quickSelect(nums, k, high + 1, right);
        }
    }
    int findKthLargest(vector<int>& nums, int k) {
        int n = nums.size();
        k = n-k;
        return quickSelect(nums,k,0,n-1);
    }
};
```

Time Complexity: O(n)
Space Complexity : O(logn) (average), O(n) (worst case)


Edit Distance

Difficulty: **Hard** Accuracy: **35.14%** Submissions: **223K+** Points: **8**

Given two strings **s1** and **s2.** Return the minimum number of operations required to convert **s1** to **s2**.
The possible operations are permitted:

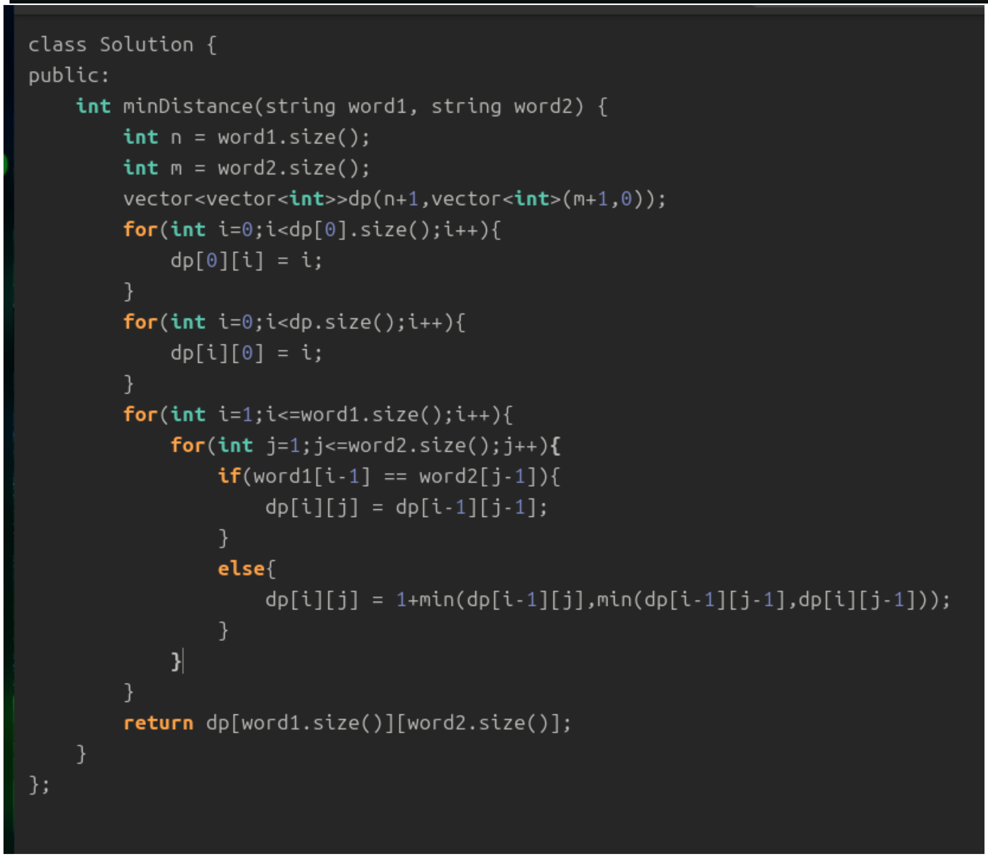
1. Insert a character at any position of the string.
2. Remove any character from the string.
3. Replace any character from the string with any other character.

Examples:

Input: s1 = "geek", s2 = "gesek"
Output: 1
Explanation: One operation is required, inserting 's' between two 'e'.

Input : s1 = "gfg", s2 = "gfg"
Output: 0
Explanation: Both strings are same.

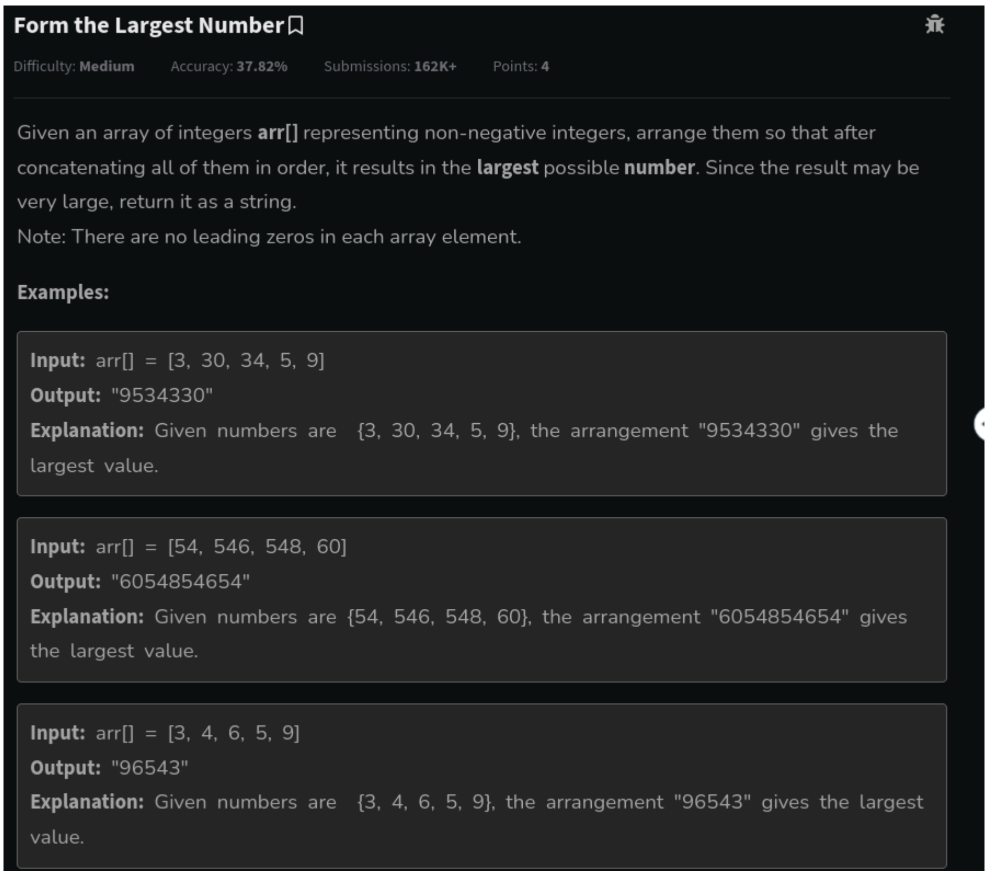
Input : s1 = "abc", s2 = "def"
Output: 3
Explanation: All characters need to be replaced to convert str1 to str2, requiring 3 replacement operations.



```
class Solution {
public:
    int minDistance(string word1, string word2) {
        int n = word1.size();
        int m = word2.size();
        vector<vector<int>>dp(n+1,vector<int>(m+1,0));
        for(int i=0;i<dp[0].size();i++){
            dp[0][i] = i;
        }
        for(int i=0;i<dp.size();i++){
            dp[i][0] = i;
        }
        for(int i=1;i<=word1.size();i++){
            for(int j=1;j<=word2.size();j++){
                if(word1[i-1] == word2[j-1]){
                    dp[i][j] = dp[i-1][j-1];
                }
                else{
                    dp[i][j] = 1+min(dp[i-1][j],min(dp[i-1][j-1],dp[i][j-1]));
                }
            }
        }
        return dp[word1.size()][word2.size()];
    }
};
```

Time Complexity : O(n×m)

Space Complexity: O(n×m)



Form the Largest Number

Difficulty: **Medium** Accuracy: **37.82%** Submissions: **162K+** Points: **4**

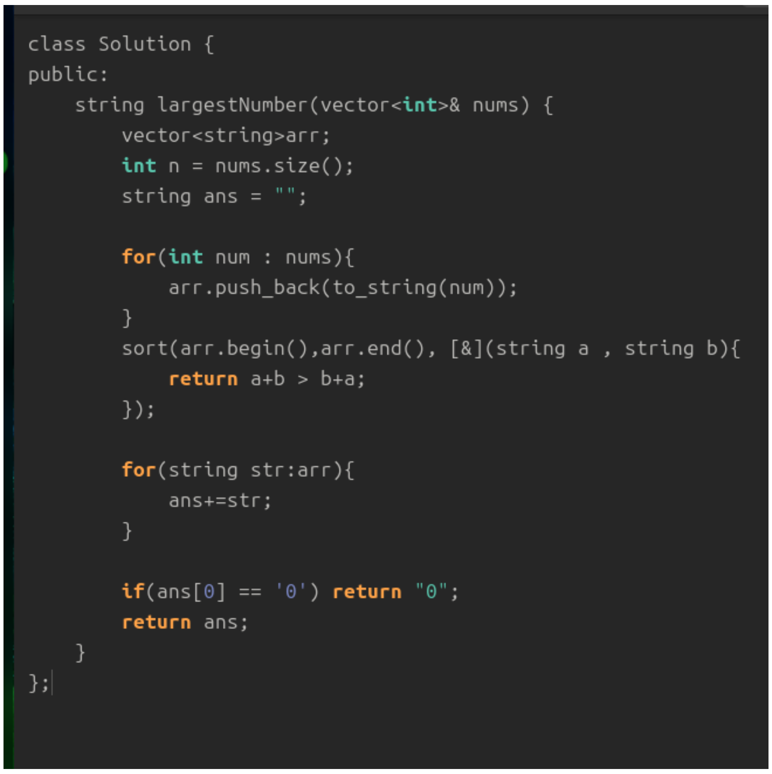
Given an array of integers **arr[]** representing non-negative integers, arrange them so that after concatenating all of them in order, it results in the **largest** possible **number**. Since the result may be very large, return it as a string.
Note: There are no leading zeros in each array element.

Examples:

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

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

```
Input: arr[] = [3, 4, 6, 5, 9]
Output: "96543"
Explanation: Given numbers are  {3, 4, 6, 5, 9}, the arrangement "96543" gives the largest value.
```



```cpp
class Solution {
public:
    string largestNumber(vector<int>& nums) {
        vector<string>arr;
        int n = nums.size();
        string ans = "";

        for(int num : nums){
            arr.push_back(to_string(num));
        }
        sort(arr.begin(),arr.end(), [&](string a , string b){
            return a+b > b+a;
        });

        for(string str:arr){
            ans+=str;
        }

        if(ans[0] == '0') return "0";
        return ans;
    }
};
```

Time Complexity:O(n·m·logn)
Space Complexity:O(n·m)