Non Repeating Character

Difficulty: **Easy** Accuracy: **40.43%** Submissions: **230K+** Points: **2**

Given a string **s** consisting of **lowercase** Latin Letters. Return the first non-repeating character in **s**. If there is no non-repeating character, return **'$'.**

Note: When you return '$' driver code will output -1.

Examples:

Input: s = "geeksforgeeks"
Output: 'f'
Explanation: In the given string, 'f' is the first character in the string which does not repeat.

Input: s = "racecar"
Output: 'e'
Explanation: In the given string, 'e' is the only character in the string which does not repeat.

Code:

```
char nonRepeatingChar(string &s) {
        unordered_map<char,int> hash;
        for(auto a:s){
            hash[a]++;
        }
        for(auto a:s){
            if(hash[a]==1){
                return a;
            }
        }
        return '$';
}
```

TimeComplexity:O(n)
SpaceComplexity:O(n)

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.

Code:

```
void bubbleSort(vector<int>& arr) {
        for(int i=arr.size()-1;i>=0;i--){
            bool flag=false;
            for(int j=0;j<i;j++){
                if(arr[j]>arr[j+1]){
                    swap(arr[j],arr[j+1]);
                    flag=true;
                }
            }
          if(flag==false)break;
        }
}
```

TimeComplexity:O(n)
SpaceComplexity: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.

Code:

```
vector<int> kLargest(vector<int>& arr, int k) {
        sort(arr.rbegin(),arr.rend());
        vector<int> ans;
        for(int i=0;i<arr.size();i++){
            if(i==k)break;
            ans.push_back(arr[i]);
        }
        return ans;
}
```

TimeComplexity:O(nlogn)
SpaceComplexity:O(n) where n is k

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.

Code:

```
int partition(vector<int>& arr, int low, int high) {
        int pivot=arr[low],i=low,j=high;
        while(i<j){
            while(arr[i]<=pivot and i<=high-1)i++;
            while(arr[j]>pivot and j>=low+1)j--;
            if(i<j)swap(arr[i],arr[j]);
        }
        swap(arr[low],arr[j]);
        return j;
}
void quickSort(vector<int>& arr, int low, int high) {
        if(low<high){
            int p=partition(arr,low,high);
            quickSort(arr,low,p-1);
            quickSort(arr,p+1,high);
        }
        return ;
}
```

TimeComplexity:O(nlogn)
SpaceComplexity:O(nlogn)

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.

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.

Code:

```
static bool cmp(int first,int second){
        string fs=to_string(first);
        string sf=to_string(second);
        // for(int i=0;i<sf.size();i++){
        //     if(fs[i]>sf[i]){
        //         return true;
        //     }
        //     else if(fs[i]<sf[i]){
        //         return false;
        //     }
        // }
        return fs+sf > sf+fs;
}
string printLargest(vector<int> &arr) {
        sort(arr.begin(),arr.end(),cmp);
        string ans="";
        for(auto a:arr){
            ans+=to_string(a);
        }
        return ans;
}
```

TimeComplexity:O(nlogn)
SpaceComplexity:O(n)