

1. Bubble Sort

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.

```
class Solution {
public:
    // Function to sort the array using bubble sort algorithm.
    void bubbleSort(vector<int>& arr) {
        int n=arr.size();
        for(int i=0;i<n-1;i++){
            for(int j=0;j<n-i-1;j++){
                if(arr[j]>arr[j+1]) swap(arr[j],arr[j+1]);
            }
        }
    }
};
```

Compilation Completed

For Input:  

4 1 3 9 7

Your Output:

1 3 4 7 9

Expected Output:

1 3 4 7 9

Time complexity : $O(n^2)$

Space complexity : $O(1)$

2.Quick Sort

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.

```
class Solution {
public:
    // Function to sort an array using quick sort algorithm.
    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;
    }

public:
    // Function that takes last element as pivot, places the pivot element at
    // its correct position in sorted array, and places all smaller elements
    // to left of pivot and all greater elements to right of pivot.
    int partition(vector<int>& arr, int l, int h) {
        int pivot = arr[h]; //? always last element
        int i = l - 1;
        for (int j = l; j <= h - 1; j++){
            if (arr[j] < pivot){
                ++i;
                swap(arr[i], arr[j]);
            }
        }
        swap(arr[i + 1], arr[h]);
        return (i + 1);
    }
};
```

Compilation Completed

For Input:  

4 1 3 9 7

Your Output:

1 3 4 7 9

Expected Output:

1 3 4 7 9

Time complexity : $O(n \log(n))$

Space complexity : $O(1)$

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.



Input: *s* = "aabbccc"

Output: '\$'

Explanation: All the characters in the given string are repeating.

```
class Solution {
public:
    // Function to find the first non-repeating character in
    char nonRepeatingChar(string &s) {
        vector<int> freq(26,0);
        vector<bool> map(26,true);
        for(auto i:s){
            freq[i-'a']++;
            if(freq[i-'a']>1) map[i-'a']=false;
        }
        for(auto i:s){
            if(map[i-'a'] && freq[i-'a']==1) return i;
        }
        return '$';
    }
};
// } Driver Code Ends
```

Compilation Completed

For Input:  

geeksforgeeks

Your Output:

f

Expected Output:

f

Time complexity : $O(n)$

Space complexity : $O(1)$

4.Edit Distance

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 helper(int m,int n,string s1,string s2,vector<vector<int>>& dp){
        if(m==0) return n;
        if(n==0) return m;
        if(dp[m][n]!=-1) return dp[m][n];
        if(s1[m-1]==s2[n-1]){
            return dp[m][n]=helper(m-1,n-1,s1,s2,dp);
        }

        int insert=1+helper(m,n-1,s1,s2,dp);
        int replace=1+helper(m-1,n-1,s1,s2,dp);
        int del=1+helper(m-1,n,s1,s2,dp);
        return dp[m][n]=min(insert,min(replace,del));
    }
    int editDistance(string s1, string s2) {
        int m=s1.size();
        int n=s2.size();
        vector<vector<int>> dp(m+1,vector<int> (n+1,-1));
        return helper(m,n,s1,s2,dp);
    }
};
// } Driver Code Ends

```

Compilation Completed

For Input:  

geek

gesek

Your Output:

1

Expected Output:

1

Time complexity : $O(n*m)$

Space complexity : $O(n*m)$

5.K Largest Number in Array

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.

```
class Solution {
public:
    vector<int> kLargest(vector<int>& arr, int k) {
        vector<int> ans;
        priority_queue<int, vector<int>, greater<int>> pq;
        for(auto i:arr){
            pq.push(i);
            while(!pq.empty() && pq.size()>k) pq.pop();
        }
        while(!pq.empty()){
            ans.push_back(pq.top());
            pq.pop();
        }
        reverse(ans.begin(),ans.end());
        return ans;
    }
};
// } Driver Code Ends
```


Compilation Completed

For Input:  

12 5 787 1 23

2

Your Output:

787 23

Expected Output:

787 23

Time complexity : $O(n \log(n))$

Space complexity : $O(n)$