Harshitha R AIDS A 22AD041 18/11/24

1. **Bubble Sort**

Difficulty: EasyAccuracy: 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() <= 103  
1 <= arr[i] <= 103

**Code**

class Solution {

public int[] bubbleSort(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

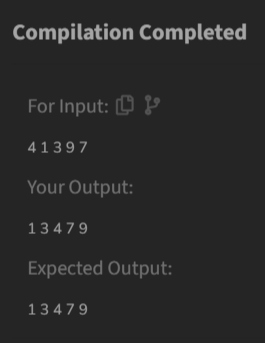
arr[j] = arr[j + 1];

arr[j + 1] = temp;

}}}

return arr;

}}

**Output**

**Time Complexity**

O(n²)

1. **Quick Sort**

Difficulty: MediumAccuracy: 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() <= 103  
1 <= arr[i] <= 104

**Code**

class Solution {

public int partition(int[] arr, int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

public void quickSort(int[] arr, int low, int high) {

if (low < high) {

int pivotIndex = partition(arr, low, high);

quickSort(arr, low, pivotIndex - 1);

quickSort(arr, pivotIndex + 1, high);

}

}

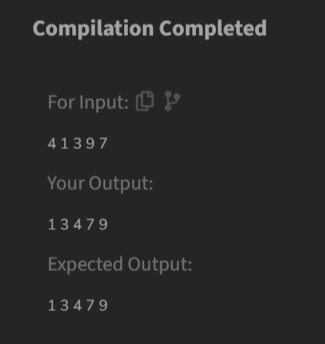
public int[] sortArray(int[] arr) {

quickSort(arr, 0, arr.length - 1);

return arr;

}

}

**Output**

**Time Complexity**

O(n log n)

1. **Non Repeating Character**

Difficulty: EasyAccuracy: 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.

Input: s = "aabbccc"

Output: '$'

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

Constraints:  
1 <= s.size() <= 105

**Code**

class Solution {

public char nonRepeatingChar(String s) {

int[] charCount = new int[26];

for (char c : s.toCharArray()) {

charCount[c - 'a']++;

}

for (char c : s.toCharArray()) {

if (charCount[c - 'a'] == 1) {

return c;

}

}

return '$';

}}

**Output**

**Time Complexity**

O(n)

1. **Edit Distance**

Difficulty: HardAccuracy: 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.

Constraints:  
1 ≤ s1.length(), s2.length() ≤ 500  
both the strings are in lowercase.

**Code**

class Solution {

public int editDistance(String s1, String s2) {

int m = s1.length();

int n = s2.length();

int[][] dp = new int[m+1][n+1];

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

dp[i][0] = i;

for (int j = 0; j <= n; j++)

dp[0][j] = j;

for (int i = 1; i <= m; i++) {

for (int j = 1; j <= n; j++) {

if (s1.charAt(i-1) == s2.charAt(j-1))

dp[i][j] = dp[i-1][j-1];

else

dp[i][j] = 1 + Math.min(dp[i-1][j],

Math.min(dp[i][j-1],

dp[i-1][j-1]));

}

}

return dp[m][n];

}

}

**Output**

**Time Complexity**

O(m\*n)

**5) k largest elements**

Difficulty: MediumAccuracy: 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 ≤ k ≤ arr.size() ≤ 106  
1 ≤ arr[i] ≤ 106

**Code**

import java.util.\*;

class Solution {

public List<Integer> kLargest(int[] arr, int k) {

PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());

for (int num : arr) {

maxHeap.offer(num);

}

List<Integer> result = new ArrayList<>();

for (int i = 0; i < k; i++) {

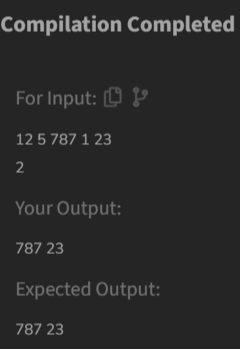
result.add(maxHeap.poll());

}

return result;

}

}

**Output**

**Time Complexity**

O(n log n)

**6) Form the Largest Number**

Difficulty: MediumAccuracy: 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.

Constraints:  
1 ≤ arr.size() ≤ 105  
0 ≤ arr[i] ≤ 105  
The sum of all the elements of the array is greater than 0.

**Code**

class Solution {

public String printLargest(int[] arr) {

String[] strArr = new String[arr.length];

for (int i = 0; i < arr.length; i++) {

strArr[i] = String.valueOf(arr[i]);

}

Arrays.sort(strArr, (a, b) -> {

String order1 = a + b;

String order2 = b + a;

return order2.compareTo(order1);

});

if (strArr[0].equals("0")) return "0";

StringBuilder result = new StringBuilder();

for (String s : strArr) {

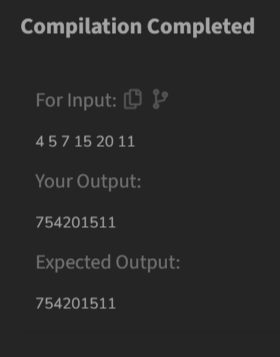
result.append(s);

}

return result.toString();

}

}

**Output**

**Time Complexity**

O(n log n)