Gopika shree M

AI&DS

22AD034

1)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

import java.util.\*;

public class Main {

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

int n = arr.length;

boolean swapped = false;

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

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

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

int temp = arr[j];

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

arr[j+1] = temp;

swapped = true;

}

}

if (!swapped){

break;

}

}

return arr;

}

public static void main(String args[]) {

Main obj = new Main();

int[] res = obj.bubblesort(new int[] {4,2,1,5,6,7,9,9});

System.***out***.println(Arrays.*toString*(res));

}

}

2) 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

import java.util.\*;

public class Main {

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

if (low < high) {

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

quicksort(arr, low, res-1);

quicksort(arr, res+1, high);

}

return arr;

}

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 static void main(String args[]) {

Main obj = new Main();

int[] res = obj.quicksort(new int[] {4,2,1,5,6,7,9,9}, 0, 7);

System.***out***.println(Arrays.*toString*(res));

}

}

3) 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

import java.util.\*;

public class Main {

public char nonRepeatingCharacter (String arr) {

HashMap<Character,Integer> set = new HashMap<>();

int n = arr.length();

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

char ch = arr.charAt(i);

set.put(ch, set.getOrDefault(ch, 0)+1);

}

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

char ch = arr.charAt(i);

if (set.get(ch) == 1) {

return ch;

}

}

return ' ';

}

public static void main(String args[]) {

Main obj = new Main();

char res = obj.nonRepeatingCharacter("aabb");

System.***out***.println(res);

}

}

4) 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.

import java.util.\*;

public class Main {

public int editDistance (String s1, String s2){

int m = s1.length();

int n = s2.length();

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

dp[0][0] = 0;

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

dp[i][0] = i;

}

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

dp[0][j] = j;

}

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

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

if (s1.charAt(i-1) == s2.charAt(j-1)) dp[i][j] = dp[i-1][j-1];

else dp[i][j] = Math.*min*(dp[i-1][j-1], Math.*min*(dp[i][j-1], dp[i-1][j]))+1;

}

}

return dp[m][n];

}

public static void main(String args[]) {

Main obj = new Main();

int res = obj.editDistance("intention", "execution");

System.***out***.println(res);

}

}

5) 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

import java.util.\*;

public class Main {

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

Arrays.*sort*(arr);

reverse(arr);

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

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

result.add(arr[i]);

}

return result;

}

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

int start = 0;

int end = arr.length -1;

while (start < end) {

int temp = arr[start];

arr[start] = arr[end];

arr[end] = arr[start];

start++;

end--;

}

return arr;

}

public static void main(String args[]) {

Main obj = new Main();

List<Integer> res = obj.kLargestElements(new int[]{45,67,1,2,3},3);

System.***out***.println(res);

}

}

6) 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

import java.util.\*;

public class Main {

public String printLargest(int[] arr) {

int n = arr.length;

String[] strArr = new String[n];

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

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

}

Arrays.*sort*(strArr, (a, b) -> (b + a).compareTo(a + b));

StringBuilder res = new StringBuilder();

for (String s : strArr) {

res.append(s);

}

if (res.charAt(0) == '0') {

return "0";

}

return res.toString();

}

public static void main(String args[]) {

Main obj = new Main();

String res = obj.printLargest(new int[]{45,67,1,2,3});

System.***out***.println(res);

}

}