**Date**: 18/11/2024

**DSA Practice Problems**

1. **Bubble Sort**

class Solution {

public static void 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;

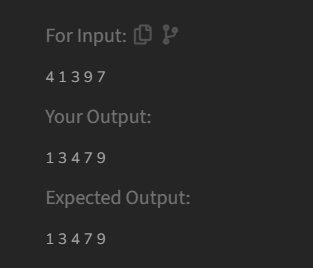
}

}

}

}

}



**Time Complexity:** O(n^2)

1. **Quick Sort**

class Solution {

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

if (low < high) {

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

quickSort(arr, low, index - 1);

quickSort(arr, index + 1, high);

}

}

static 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];

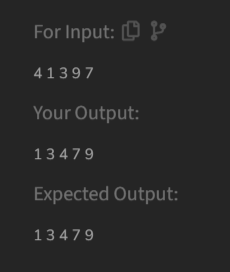
arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

}



**Time Complexity:** O (n log n)

1. **NonRepeating Character**

class Solution {

static char nonRepeatingChar(String s) {

HashMap<Character, Integer> HashMap1 = new HashMap<>();

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

if (HashMap1.containsKey(ch)) {

HashMap1.put(ch, HashMap1.get(ch) + 1);

} else {

HashMap1.put(ch, 1);

}

}

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

if (HashMap1.get(ch) == 1) {

return ch;

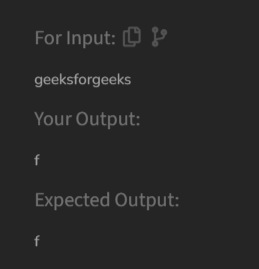
}

}

return '$';

}

}



**Time Complexity:** O(n)

1. **Edit Distance**

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]));

}

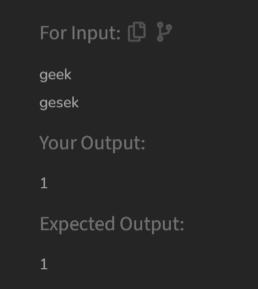
}

}

return dp[m][n];

}

}



**Time Complexity:** O(m\*n)

1. **K Largest Elements**

class Solution {

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

PriorityQueue<Integer> minheap = new PriorityQueue<>();

for (int num : arr) {

minheap.add(num);

if (minheap.size() > k) {

minheap.poll();

}

}

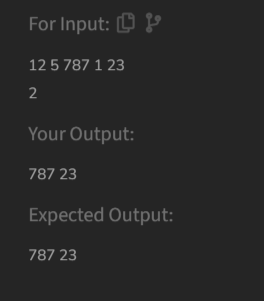
List<Integer> result = new ArrayList<>(minheap);

result.sort(Collections.reverseOrder());

return result;

}

}



**Time Complexity**: O(n log k)

1. **Form the Largest Number**

class Solution {

String printLargest(int[] arr) {

String[] numbers = Arrays.stream(arr).mapToObj(String::valueOf).toArray(String[]::new);

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

String result = String.join("", numbers);

if (result.startsWith("0")) {

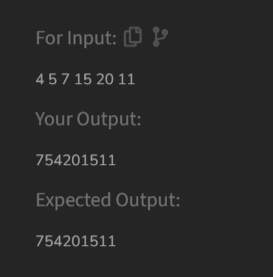
return "0";

}

return result;

}

}



**Time Complexity:** O (n log n)