

# DSA Practice – 6

## 1. Bubble Sort

The screenshot shows a coding platform interface with a problem solved successfully. The left sidebar displays the following statistics:

- Test Cases Passed: 1115 / 1115
- Attempts: Correct / Total: 1 / 1
- Accuracy: 100%
- Points Scored: 2 / 2
- Time Taken: 0.59
- Your Total Score: 49 ↑

The main editor shows the following Java code for the Bubble Sort algorithm:

```
1 // } Driver Code Ends
9 class Solution {
10     public static void bubbleSort(int arr[]) {
11         int n = arr.length;
12         for (int i = 0; i < n - 1; i++) {
13             boolean swapped = false;
14             for (int j = 0; j < n - i - 1; j++) {
15                 if (arr[j] > arr[j + 1]) {
16                     int temp = arr[j];
17                     arr[j] = arr[j + 1];
18                     arr[j + 1] = temp;
19                     swapped = true;
20                 }
21             }
22             if (!swapped) break;
23         }
24     }
25 }
26 // } Driver Code Ends
```

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

## 2. Quick Sort

The screenshot shows a coding platform interface with a problem solved successfully. The left sidebar displays the following statistics:

- Test Cases Passed: 1120 / 1120
- Attempts: Correct / Total: 1 / 1
- Accuracy: 100%
- Points Scored: 4 / 4
- Time Taken: 0.54
- Your Total Score: 53 ↑

The main editor shows the following Java code for the Quick Sort algorithm:

```
1 // } Driver Code Ends
30 class Solution {
31     static void quickSort(int arr[], int low, int high) {
32         if (low < high) {
33             int pivotIndex = partition(arr, low, high);
34             quickSort(arr, low, pivotIndex - 1);
35             quickSort(arr, pivotIndex + 1, high);
36         }
37     }
38     static int partition(int arr[], int low, int high) {
39         int pivot = arr[high];
40         int i = low - 1;
41         for (int j = low; j < high; j++) {
42             if (arr[j] <= pivot) {
43                 i++;
44                 int temp = arr[i];
45                 arr[i] = arr[j];
46                 arr[j] = temp;
47             }
48         }
49         int temp = arr[i + 1];
50         arr[i + 1] = arr[high];
51         arr[high] = temp;
52         return i + 1;
53     }
54 }
55 }
```

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

### 3.Non Repeating Character

The screenshot displays a coding platform interface with a green header bar containing navigation links: Courses, Tutorials, Jobs, Practice, and Contests. The main area is divided into two panels. The left panel, titled 'Output Window', shows 'Compilation Results' for a problem solved successfully. It reports 'Test Cases Passed: 1130 / 1130', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 2 / 2', and 'Time Taken: 0.35'. The right panel shows the Java code for the solution, which uses a frequency array to find the first non-repeating character in a string.

```
1 // } Driver Code Ends
29 class Solution {
30     static char nonRepeatingChar(String s) {
31         int[] freq = new int[26];
32         for (int i = 0; i < s.length(); i++) {
33             freq[s.charAt(i) - 'a']++;
34         }
35         for (int i = 0; i < s.length(); i++) {
36             if (freq[s.charAt(i) - 'a'] == 1) {
37                 return s.charAt(i);
38             }
39         }
40         return '$';
41     }
42 }
43 // } Driver Code Ends
```

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

### 4.K Largest Element

The screenshot displays a coding platform interface with a green header bar containing navigation links: Courses, Tutorials, Jobs, Practice, and Contests. The main area is divided into two panels. The left panel, titled 'Output Window', shows 'Compilation Results' for a problem solved successfully. It reports 'Test Cases Passed: 1111 / 1111', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Time Taken: 0.74'. The right panel shows the Java code for the solution, which uses a max heap to find the k largest elements in an array.

```
1 // } Driver Code Ends
9 class Solution {
10     static ArrayList<Integer> kLargest(int arr[], int k) {
11         PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
12         for (int num : arr) {
13             maxHeap.add(num);
14         }
15         ArrayList<Integer> result = new ArrayList<>();
16         for (int i = 0; i < k; i++) {
17             result.add(maxHeap.poll());
18         }
19         return result;
20     }
21 }
22 // } Driver Code Ends
```

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

## 5. Form the Largest Element

The screenshot shows a coding platform interface with a navigation bar at the top containing 'Courses', 'Tutorials', 'Jobs', 'Practice', and 'Contests'. The main area is divided into two panels. The left panel, titled 'Output Window', displays 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1111 / 1111', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Time Taken: 1.25'. The right panel shows the Java code for the solution. The code defines a class 'Solution' with a method 'printLargest' that sorts an array of strings and concatenates them to form the largest possible number. The code is as follows:

```
1 // Driver Code Ends
24 class Solution {
25     String printLargest(String[] arr) {
26         Arrays.sort(arr, (a, b) -> {
27             String ab = a + b;
28             String ba = b + a;
29             return ba.compareTo(ab);
30         });
31         if (arr[0].equals("0")) {
32             return "0";
33         }
34         StringBuilder result = new StringBuilder();
35         for (String num : arr) {
36             result.append(num);
37         }
38         return result.toString();
39     }
40 }
41
```

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

## 6. Edit distance

The screenshot shows a coding platform interface with a navigation bar at the top containing 'Courses', 'Tutorials', 'Jobs', 'Practice', and 'Contests'. The main area is divided into two panels. The left panel, titled 'Output Window', displays 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1115 / 1115', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 8 / 8', and 'Time Taken: 0.2'. The right panel shows the Java code for the solution. The code defines a class 'Solution' with a method 'editDistance' that uses dynamic programming to calculate the minimum number of operations (insert, delete, replace) required to convert one string into another. The code is as follows:

```
1 // Driver Code Ends
22 class Solution {
23     public int editDistance(String s1, String s2) {
24         int m = s1.length();
25         int n = s2.length();
26         int[][] dp = new int[m + 1][n + 1];
27         for (int i = 0; i <= m; i++) {
28             dp[i][0] = i;
29         }
30         for (int j = 0; j <= n; j++) {
31             dp[0][j] = j;
32         }
33         for (int i = 1; i <= m; i++) {
34             for (int j = 1; j <= n; j++) {
35                 if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
36                     dp[i][j] = dp[i - 1][j - 1];
37                 } else {
38                     dp[i][j] = 1 + Math.min(Math.min(dp[i - 1][j],
39                                                         dp[i][j - 1]),
40                                                         dp[i - 1][j - 1]);
41                 }
42             }
43         }
44         return dp[m][n];
45     }
46 }
47
```

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