

# DSA Practice - 6

NAME : SIDHARTH.N

DEP : AI&DS - 'C'

## Bubble sort

The screenshot shows a coding platform interface for the Bubble Sort problem. The left panel, titled 'Output Window', displays the following information:

- Compilation Results:** Custom Input, Y.O.G.I. (AI Bot)
- Problem Solved Successfully:** ✓
- Test Cases Passed:** 1115 / 1115
- Attempts:** Correct / Total: 1 / 2
- Accuracy:** 50%
- Points Scored:** 2 / 2
- Time Taken:** 0.57
- Your Total Score:** 14 ↑
- Solve Next:** Selection Sort, Insertion Sort, Counting Sort
- Kick start your career with GfG 160!**

The right panel shows the Java code for the bubble sort algorithm:

```
1 // Driver Code Starts
9 // User function Template for Java
11 class Solution {
12 // Function to sort the array using bubble sort algorithm.
13 public static void bubbleSort(int arr[]) {
14 // code here
15 for(int i = 0; i<arr.length-1;i++){
16     boolean flag = false;
17     for(int j = 0; j < arr.length-i-1; j++){
18         if(arr[j]>arr[j+1]){
19             flag = true;
20             int temp = arr[j];
21             arr[j] = arr[j+1];
22             arr[j+1] = temp;
23         }
24     }
25     if(!flag){
26         break;
27     }
28 }
29 }
30 }
31 // Driver Code Ends
```

Time Complexity :  $O(n^2)$

## 2. Non Repeating Character

The screenshot shows a coding platform interface for the Non Repeating Character problem. The left panel, titled 'Output Window', displays the following information:

- Compilation Results:** Custom Input, Y.O.G.I. (AI Bot)
- Problem Solved Successfully:** ✓
- Test Cases Passed:** 1130 / 1130
- Attempts:** Correct / Total: 1 / 1
- Accuracy:** 100%
- Points Scored:** 2 / 2
- Time Taken:** 0.5
- Your Total Score:** 16 ↑
- Solve Next:** Reverse Words, Longest substring with distinct characters, Balloon Everywhere
- Kick start your career with GfG 160!**

The right panel shows the Java code for finding the first non-repeating character in a string:

```
1 // Driver Code Starts
29 // User function Template for Java
31
32
33 class Solution {
34 // Function to find the first non-repeating character in a string.
35 static char nonRepeatingChar(String s) {
36 // Your code here
37 HashMap<Character,Integer> map = new HashMap<>();
38 for(char j : s.toCharArray()){
39     map.put(j,map.getOrDefault(j,0)+1);
40 }
41 for(char i : s.toCharArray()){
42     if(map.get(i) == 1){
43         return i;
44     }
45 }
46 return '$';
47 }
48 }
49
```

Time Complexity :  $O(n)$

### 3.k largest elements

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully [Suggest Feedback](#)

Test Cases Passed: 1111 / 1111

Attempts: Correct / Total: 1 / 3

Accuracy: 33%

Points Scored: 4 / 4

Time Taken: 0.53

Your Total Score: 46

Solve Next

Merge k Sorted Arrays Maximum Sum Combination Optimal Array

Kick start your career with GfG 160!

```
1 // User function Template for Java
2
3 class Solution {
4
5     // Function to find the first negative integer in every window of size k
6     static List<Integer> kLargest(int arr[], int k) {
7         // write code here
8         PriorityQueue<Integer> q = new PriorityQueue<>();
9         List<Integer> ans = new ArrayList<>();
10        for(int i = 0; i < arr.length; i++){
11            q.offer(arr[i]);
12            if(q.size() > k){
13                q.poll();
14            }
15        }
16        while (!q.isEmpty()){
17            ans.add(q.poll());
18        }
19        Collections.reverse(ans);
20        return ans;
21    }
22 }
23
24 // } Driver Code Ends
```

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

### 4. Form the Largest Number

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully [Suggest Feedback](#)

Test Cases Passed: 1111 / 1111

Attempts: Correct / Total: 1 / 1

Accuracy: 100%

Points Scored: 4 / 4

Time Taken: 1.17

Your Total Score: 50

Solve Next

Max sum in the configuration Maximum Index Maximum Index

Kick start your career with GfG 160!

```
1 // User function Template for Java
2
3 class Solution {
4     String printLargest(int[] arr) {
5         String[] nums = Arrays.stream(arr)
6             .mapToObj(String::valueOf)
7             .toArray(String[]::new);
8
9         Arrays.sort(nums, (a, b) -> (b + a).compareTo(a + b));
10
11         if (nums[0].equals("0")) {
12             return "0";
13         }
14
15         StringBuilder result = new StringBuilder();
16         for (String num : nums) {
17             result.append(num);
18         }
19         return result.toString();
20     }
21 }
```

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

## 5. Quick Sort

The screenshot displays a coding platform interface with a dark theme. The top navigation bar includes links for Courses, Tutorials, Jobs, Practice, and Contests. The main area is divided into two panels. The left panel, titled 'Output Window', shows 'Compilation Results' with a green checkmark indicating 'Problem Solved Successfully'. It also displays performance metrics: 'Test Cases Passed: 1120 / 1120', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Time Taken: 0.75'. Below these, there are buttons for 'Merge Sort', 'Maximum Intervals Overlap', and 'Print the Sequence', along with a green banner encouraging users to 'Kick start your career with GFG 160!'. The right panel shows the Java code for the Quick Sort algorithm, which includes a `quickSort` method and a `partition` method. The code is written in Java 1.8 and is enclosed in a `Solution` class.

```
1 // Driver Code Starts
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32 class Solution {
33     // Function to sort an array using quick sort algorithm.
34     static void quickSort(int arr[], int low, int high) {
35         // code here
36         if (low < high) {
37             int pi = partition(arr, low, high);
38             quickSort(arr, low, pi - 1);
39             quickSort(arr, pi + 1, high);
40         }
41     }
42
43     static int partition(int arr[], int low, int high) {
44         // your code here
45         int i = low - 1;
46         int pivot = arr[high];
47
48         for (int j = low; j < high; j++) {
49             if (arr[j] < pivot) {
50                 i++;
51                 int temp = arr[i];
52                 arr[i] = arr[j];
53                 arr[j] = temp;
54             }
55         }
56         int temp = arr[i + 1];
57         arr[i + 1] = arr[high];
58         arr[high] = temp;
59         return i + 1;
60     }
61 }
62
63
64
```

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