DSA PRACTICE QUESTIONS – DAY 6

Name: Jeevitha R **Reg No**: 22IT040 **Date**: 18/11/2024

1. Bubble Sort Code Solution:

arr[j+1]=temp; swapped=true;

if (!swapped) break;

}

Output:

```
Bubble Sort ☐
                                                                                                                       10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
                                                                                                                            class Solution {
                                                                                                                                 public static void bubbleSort(int arr[]) {
Given an array, arr[]. Sort the array using bubble sort algorithm
                                                                                                                                      int n=arr.length;
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 Window
                                                                                                                                           if (!swapped) break;
Problem Solved Successfully
                                                                                               Suggest Feedback
 1115 / 1115
                                                          Accuracy: 100%
```

Time complexity: $O(n^2)$ Space Complexity: O(1)

2. Quick Sort

```
Code Solution:
```

```
class Solution {
  static 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);}
  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) {</pre>
          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;
}
```

Output:

```
Quick Sort
                                                                                                                                 32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
61
62
63
64
66
66
66
                                                                                                                                       class Solution {
                                                                                                                                             static void quickSort(int arr[], int low, int high) {
 Difficulty: Medium
                                                                                                                                                       if (low<high) {
int pivotIndex=partition(arr, low, high);</pre>
 Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, arr[] in ascending order.
                                                                                                                                                       quickSort(arr, low, pivotIndex-1);
quickSort(arr, pivotIndex+1, high);
 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.
                                                                                                                                             static int partition(int arr[], int low, int high) {
                                                                                                                                                  int pivot=arr[high];
int i=low-1;
 Examples:
                                                                                                                                                  for (int j=low; j<high; j++) {
    if (arr[j]<pivot) {</pre>
Output Window
Problem Solved Successfully
                                                                                                        Suggest Feedback
 1120 / 1120
                                                               1/1
                                                               Accuracy : 100%
```

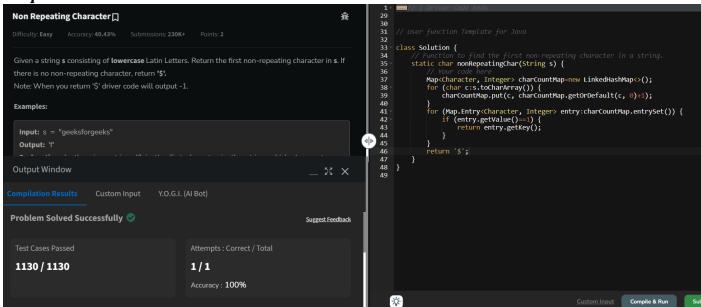
Time Complexity: O (n²)
Space Complexity: O (logn)

3. Non-Repeating Characters

```
Code Solution:
```

```
class Solution {
    // Function to find the first non-repeating character in a string.
    static char nonRepeatingChar(String s) {
        // Your code here
        Map<Character, Integer> map=new LinkedHashMap<>();
        for (char c:s.toCharArray()) {
            map.put(c, map.getOrDefault(c, 0)+1);
        }
        for (Map.Entry<Character, Integer> entry:map.entrySet()) {
            if (entry.getValue()==1) {
                return entry.getKey();
            }
        }
        return '$';
    }
}
```

Output:



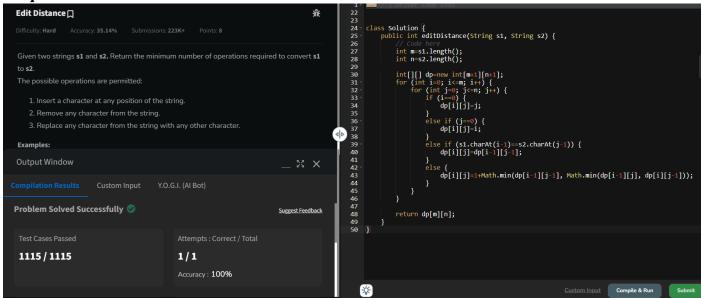
Time complexity: O (n)
Space Complexity: O (n)

4. Edit Distance

Code Solution:

```
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++) {
       for (int j=0; j <= n; j++) {
          if(i==0) {
             dp[i][j]=j;
          else if (j==0) {
             dp[i][j]=i;
          else 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-1], Math.min(dp[i-1][j], dp[i][j-1]));
        }
     return dp[m][n];
}
```

Output:



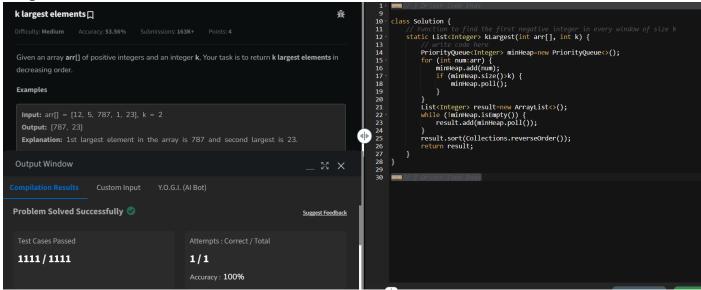
Time Complexity: O (m*n)
Space Complexity: O (m*n)

5. k Largest Element

Code Solution:

```
class Solution {
    // Function to find the first negative integer in every window of size k
    static List<Integer> kLargest(int arr[], int k) {
        // write code here
        PriorityQueue<Integer> minHeap=new PriorityQueue<>();
        for (int num:arr) {
            minHeap.add(num);
            if (minHeap.size()>k) {
                 minHeap.poll();
            }
        }
        List<Integer> result=new ArrayList<>();
        while (!minHeap.isEmpty()) {
            result.add(minHeap.poll());
        }
        result.sort(Collections.reverseOrder());
        return result;
    }
}
```

Output:



Time Complexity: O (n*logk)
Space Complexity: O (k)

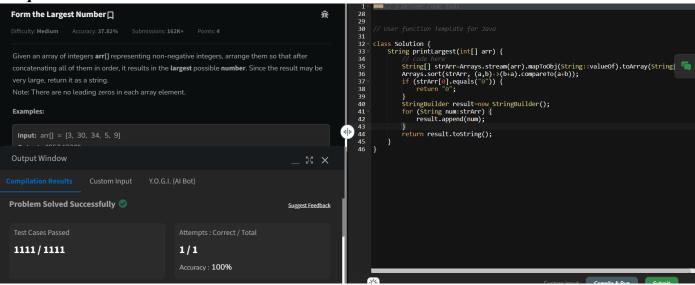
6. Form the Largest Number

Code Solution:

```
class Solution {
   String printLargest(int[] arr) {
```

```
// code here
    String[]
nums=Arrays.stream(arr).mapToObj(String::valueOf).toArray(String[]::new);
    Arrays.sort(nums, (a,b)->(b+a).compareTo(a+b));
    if (nums[0].equals("0")) {
        return "0";
    }
    StringBuilder result=new StringBuilder();
    for (String num:nums) {
        result.append(num);
    }
    return result.toString();
}
```

Output:



Time Complexity: O (n*klogn)
Space Complexity: O (n)