DSA Practice Problems

Set - 6

1. **Bubble Sort:**

Given an array, **arr[]**. Sort the array using bubble sort algorithm.

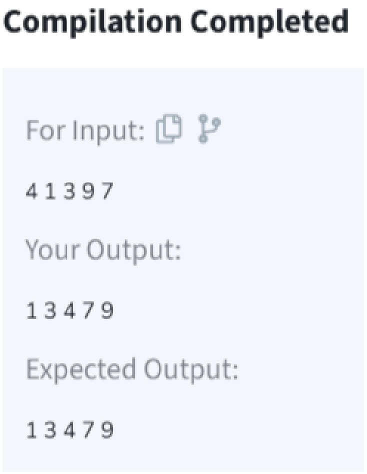
Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]

Code:

```
class Solution {
    // Function to sort the array using bubble sort algorithm.
    public static void bubbleSort(int arr[]) {
        // code here;
        int temp;
        for(int i=0;i<arr.length-1;i++){
            for(int j=i+1;j<arr.length;j++){
                if(arr[i]>arr[j]){
                    temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
            }
        }
        for(int num:arr){

        }
    }
}
```



Compilation Completed

For Input:

```
4 1 3 9 7
```

Your Output:

```
1 3 4 7 9
```

Expected Output:

```
1 3 4 7 9
```

Time Complexity: O(n^2)

2. **Non Repeating Character:**

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

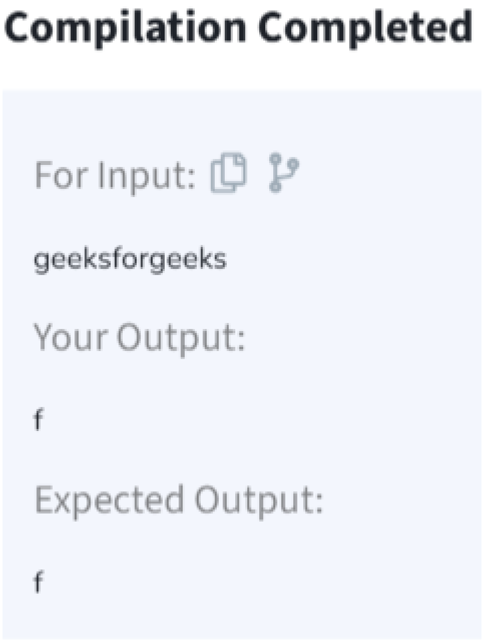
Input: s = "geeksforgeeks"

Output: 'f'

Explanation: In the given string, 'f' is the first character in the string which does not repeat.

Code:

```
class Solution {
   // Function to find the first non-repeating character in a string.
   static char nonRepeatingChar(String s) {
      int n = s.length();
      for (int i = 0; i < n; ++i) {
         boolean found = false;
         for (int j = 0; j < n; ++j) {
            if (i != j && s.charAt(i) == s.charAt(j)) {
               found = true;
               break;
            }
         }
         if (found == false)
            return s.charAt(i);
      }
      return '$';
   }
}
```



Compilation Completed

For Input:

```
geeksforgeeks
```

Your Output:

```
f
```

Expected Output:

```
f
```

Time Complexity: O(n^2)

3. **Quick Sort:**

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.

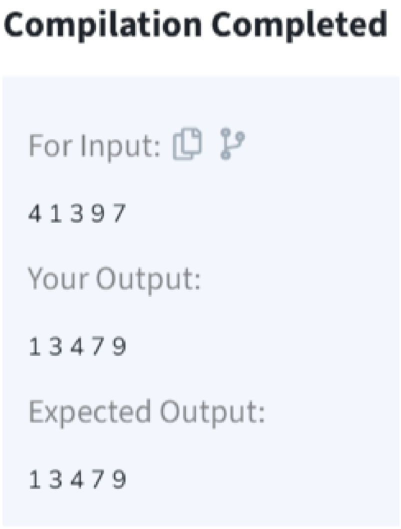
Note: The **low** and **high** are inclusive.

Input: arr[] = [4, 1, 3, 9, 7]

Output: [1, 3, 4, 7, 9]

Code:

```
class Solution {
    // Function to sort an array using quick sort algorithm.
    static void quickSort(int arr[], int low, int high) {
        // code here
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }
    static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
    static int partition(int arr[], int low, int high) {
        // your code here
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j <= high - 1; j++) {
            if (arr[j] < pivot) {
                i++;
                swap(arr, i, j);
            }
        }
        swap(arr, i + 1, high);
        return i + 1;
    }
}
```



Compilation Completed

For Input:

```
4 1 3 9 7
```

Your Output:

```
1 3 4 7 9
```

Expected Output:

```
1 3 4 7 9
```

Time Complexity: Best Case: (Ω(n log n)), Average Case (θ(n log n)), Worst Case: (O(n²))

4. **Edit Distance:**

Given two strings s1 and s2. Return the minimum number of operations required to convert s1 to s2.

The possible operations are permitted:

Insert a character at any position of the string.

Remove any character from the string.

Replace any character from the string with any other character.

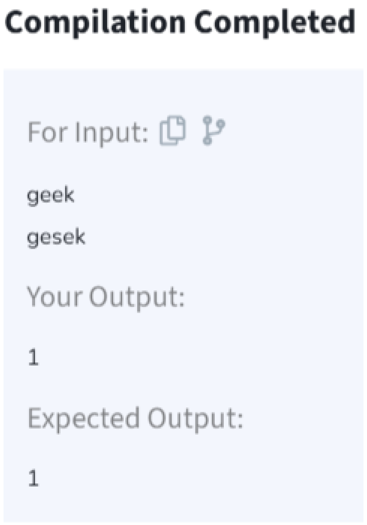
Input: s1 = "geek", s2 = "gesek"

Output: 1

Explanation: One operation is required, inserting 's' between two 'e'.

Code:

```
class Solution {
  public int editDistance(String s1, String s2) {
    int m = s1.length(), 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] = Math.min(
            dp[i - 1][j - 1], // Replace
            Math.min(dp[i][j - 1], dp[i - 1][j]) // Insert or Delete
          ) + 1;
        }
      }
    }
    return dp[m][n];
  }
}
```



Compilation Completed

For Input:

```
geek
gesek
```

Your Output:

```
1
```

Expected Output:

```
1
```

Time Complexity: O(n)

5. **K Largest Element:**

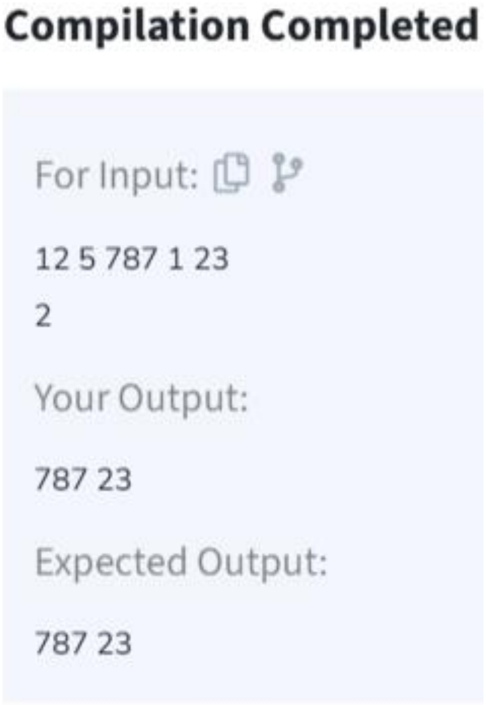
Given an array **arr[]** of positive integers and an integer **k**, Your task is to return **k largest elements** in decreasing order.

Input: arr[] = [12, 5, 787, 1, 23], k = 2

Output: [787, 23]

Code:

```
class Solution {
   // Function to find the first negative integer in every window of size k
   static List<Integer> kLargest(int arr[], int k) {
      List<Integer> list = new ArrayList<>();
      for (int num : arr) {
         list.add(num);
      }
      Collections.sort(list, Collections.reverseOrder());
      return list.subList(0, k);
   }
}
```



Compilation Completed

For Input:

```
12 5 787 1 23
2
```

Your Output:

```
787 23
```

Expected Output:

```
787 23
```

Time Complexity: O(nlogn)

6. **Form the Largest Number:**

Given an integer **N** the task is to find the largest number which is smaller or equal to it and has its digits in non-decreasing order.

Input: N = 200

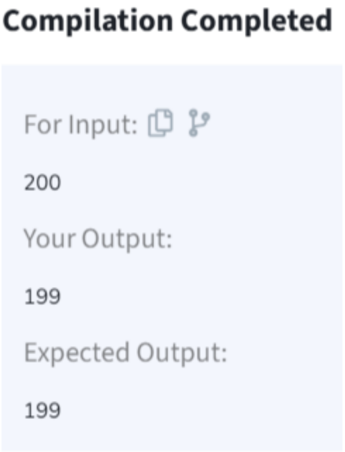
Output: 199

Explanation:

If the given number is 200, the largest number which is smaller or equal to it having digits in non-decreasing order is 199.

Code:

```
class Solution{
   static int find(int N){
      char[] digits = String.valueOf(N).toCharArray();
      int length = digits.length;
      int i;
      for (i = 1; i < length; i++) {
         if (digits[i] < digits[i - 1]) {
            break;
         }
      }
      if (i == length) {
         return N;
      }
      while (i > 0 && digits[i - 1] > digits[i]) {
         digits[i - 1]--;
         i--;
      }
      for (int j = i + 1; j < length; j++) {
         digits[j] = '9';
      }
      return Integer.parseInt(new String(digits));
   }
}
```



Compilation Completed

```
For Input:
200
Your Output:
199
Expected Output:
199
```

Time Complexity: O(n)