DSA PRACTICE PROBLEMS DATE:18/11/24

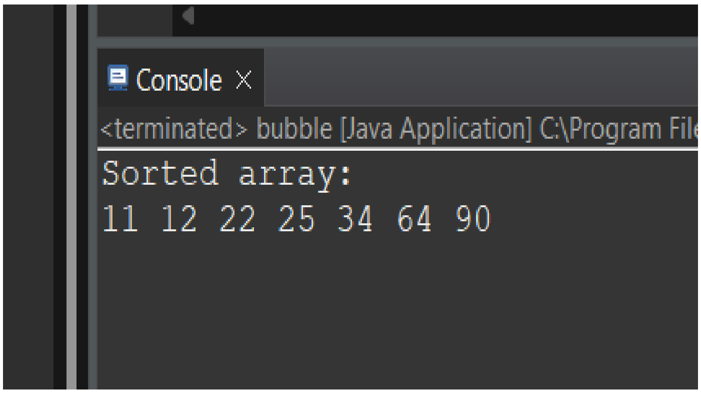
1. Given an array, **arr[]**. Sort the array using bubble sort algorithm.

Input: arr[] = [64, 34, 25, 12, 22, 11, 90];

Output: [11 12 22 25 34 64 90];

CODE:

```
class Solution {
    public static void bubbleSort(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            boolean swapped = false;
            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;
                    swapped = true;
                }
            }
            if (!swapped) break;
        }
    }
}
```



```
<terminated> bubble [Java Application] C:\Program File
Sorted array:
11 12 22 25 34 64 90
```

Time Complexity: O(n^2)

2. 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.

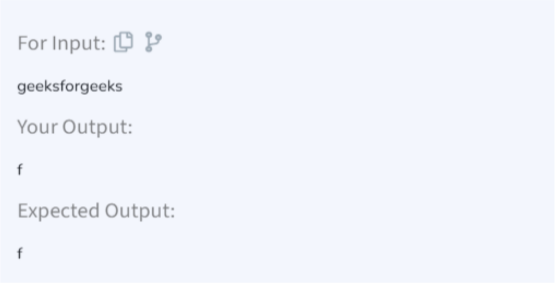
Examples:

Input: s = "geeksforgeeks"

Output: 'f'

CODE:

```
class Solution {
   static char nonRepeatingChar(String s) {
      int[] freq = new int[26];
      for (char c : s.toCharArray()) {
         freq[c - 'a']++;
      }
      for (char c : s.toCharArray()) {
         if (freq[c - 'a'] == 1) {
            return c;
         }
      }
      return '$';
   }
}
```



For Input:

geeksforgeeks

Your Output:

f

Expected Output:

f

Time Complexity:O(n)

3. Given two strings **s1** and **s2.** Return the minimum number of operations required to convert **s1** to **s2**.
The possible operations are permitted:

1. Insert a character at any position of the string.
2. Remove any character from the string.

3. Replace any character from the string with any other character.

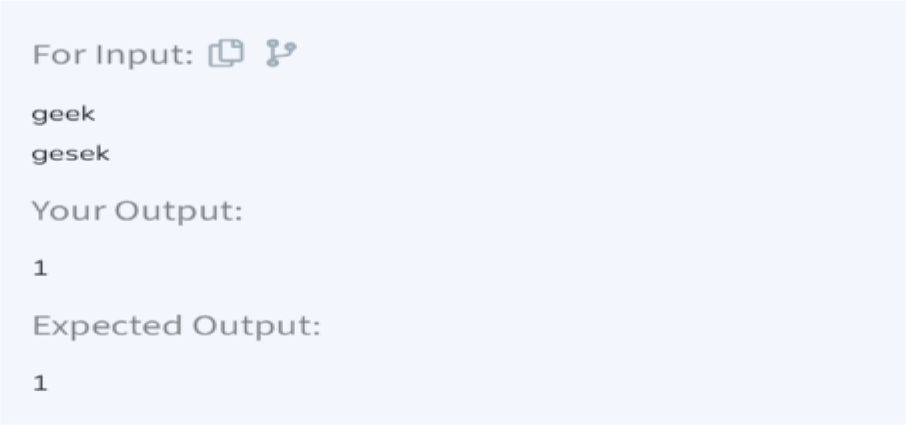
Input: s1 = "geek", s2 = "gesek"

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();
        int n = s2.length();
        int[][] dp = new int[m + 1][n + 1];
        for (int i = 0; i <= m; i++) {
            dp[i][0] = i; // All deletions
        }
        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 - 1],
                            Math.min(dp[i - 1][j],
                                     dp[i][j - 1]));
                }
            }
        }
        return dp[m][n];
    }
}
```



```
For Input:
geek
gesek
Your Output:
1
Expected Output:
1
```

Time Complexity: O(m*n)

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

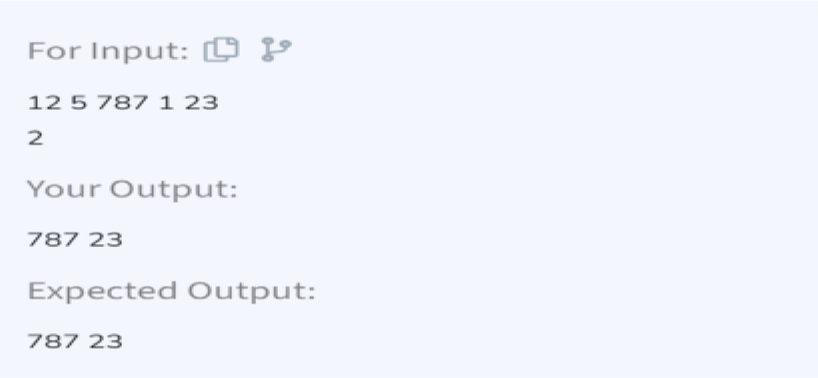
Input: arr[] = [12, 5, 787, 1, 23], k = 2

Output: [787, 23]

Explanation: 1st largest element in the array is 787 and second largest is 23.

CODE:

```
class Solution {
  static List<Integer> kLargest(int arr[], int k) {
    int n = arr.length;
    Integer[] arrInteger =
        Arrays.stream(arr).boxed().toArray(Integer[]::new);
    Arrays.sort(arrInteger, Collections.reverseOrder());
    ArrayList<Integer> res = new ArrayList<>();
    for (int i = 0; i < k; i++)
      res.add(arrInteger[i]);
    return res;
  }
}
```



For Input:

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

Your Output:

```
787 23
```

Expected Output:

```
787 23
```

Time Complexity:O(n log n)

5. Given an array of integers **arr[]** representing non-negative integers, arrange them so that after concatenating all of them in order, it results in the **largest** possible **number**. Since the result may be very large, return it as a string.

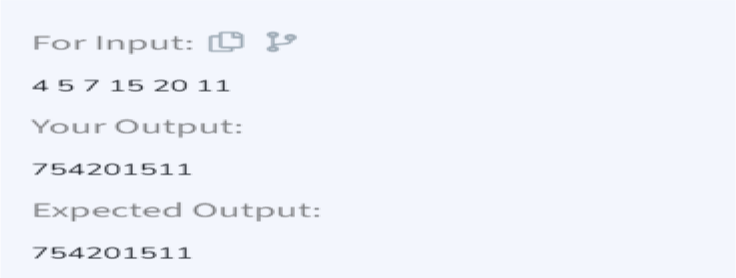
Input: arr[] = [3, 30, 34, 5, 9]

Output: "9534330"

Explanation: Given numbers are {3, 30, 34, 5, 9}, the arrangement "9534330" gives the largest value.

CODE:

```
class Solution {
   String printLargest(int[] arr) {
       String[] strArr = Arrays.stream(arr)
                         .mapToObj(String::valueOf)
                         .toArray(String[]::new);
      Arrays.sort(strArr, new Comparator<String>() {
         public int compare(String s1, String s2) {
            String order1 = s1 + s2;
            String order2 = s2 + s1;
            return order2.compareTo(order1);
         }
      });
      if (strArr[0].equals("0")) {
         return "0";
      }
      StringBuilder result = new StringBuilder();
      for (String num : strArr) {
            result.append(num);

        }

        return result.toString();

    }

}
```



For Input:

4 5 7 15 20 11

Your Output:

754201511

Expected Output:

754201511

Time Complexity: O(n log n)

6. 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.

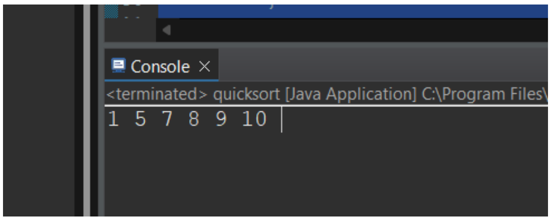
Note: The **low** and **high** are inclusive.

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

Output: [1, 3, 4, 7, 9]
Explanation: After sorting, all elements are arranged in ascending order.

CODE:

```
package util;

public class quicksort {

    static int partition(int[] arr, int low, int high) {

        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;
    }
    static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
    static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }
    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        int n = arr.length;
        quickSort(arr, 0, n - 1);
        for (int val : arr) {
            System.out.print(val + " ");
        }
    }
}
```



```
<terminated> quicksort [Java Application] C:\Program Files\
1 5 7 8 9 10
```

Time Complexity: O(n log n)