Coding Practice-6

1.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.

Examples:

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

Input: arr[] = [2, 1, 6, 10, 4, 1, 3, 9, 7]
Output: [1, 1, 2, 3, 4, 6, 7, 9, 10]
Explanation: Duplicate elements (1) are retained in sorted order.

Input: arr[] = [5, 5, 5, 5]
Output: [5, 5, 5, 5]
Explanation: All elements are identical, so the array remains unchanged.

Code:

```
class Solution {
    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 + " ");
    }
  }
}
```

Output:

Sorted Array

1 5 7 8 9 10

Time complexity: O(n)

2. Bubble Sort

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

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: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Input: arr[] = [1, 2, 3, 4, 5]
Output: [1, 2, 3, 4, 5]
Explanation: An array that is already sorted should remain unchanged after applying bubble sort.

Code:

```
import java.io.*;
class GFG {
    static void bubbleSort(int arr[], int n) {
        int i, j, temp;
        boolean swapped;
        for (i = 0; i < n - 1; i++) {
            swapped = false;
            for (j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
```

```
                }
            }
            if (swapped == false)
                break;
        }
    }
    static void printArray(int arr[], int size) {
        int i;
        for (i = 0; i < size; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
    public static void main(String args[]) {
        int arr[] = { 64, 34, 25, 12, 22, 11, 90 };
        int n = arr.length;
        bubbleSort(arr, n);
        System.out.println("Sorted array: ");
        printArray(arr, n);
    }
}
```

Output:

Sorted array:

11 12 22 25 34 64 90

Time Complexity:$O(n^2)$

3. 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.

Examples:

Input: s = "geeksforgeeks"
Output: 'f'
Explanation: In the given string, 'f' is the first character in the string which does not repeat.

Input: s = "racecar"
Output: 'e'
Explanation: In the given string, 'e' is the only character in the string which does not repeat.

Input: s = "aabbccc"
Output: '$'
Explanation: All the characters in the given string are repeating.

Code:

```
class Solution {
    static char nonRepeatingChar(String s) {
        int[] freq = new int[26];
        int n = s.length();
        for (int i = 0; i < n; i++) {
            freq[s.charAt(i) - 'a']++;
        }
        for (int i = 0; i < n; i++) {
            if (freq[s.charAt(i) - 'a'] == 1) {
                return s.charAt(i);
            }
        }
        return '$';
```

```
    }
    public static void main(String[] args) {
        String s = "hghjhjjn";
        char result = nonRepeatingChar(s);
        System.out.println("First non-repeating character: " + (result == '$' ? -1 : result));
    }
}
```

Output:

First non-repeating character: g

Time complexity: O(n)

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:

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.

Examples:

Input: s1 = "geek", s2 = "gesek"
Output: 1
Explanation: One operation is required, inserting 's' between two 'e'.

Input : s1 = "gfg", s2 = "gfg"
Output: 0
Explanation: Both strings are same.

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++) {
            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];
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        String s1 = "horse";
        String s2 = "ros";
        System.out.println("Minimum operations required: " + solution.editDistance(s1, s2));
    }
}
```

Output: Minimum operations required: 3

Time complexity: O(m×n)

5.K Largest Elements

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

Examples

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.

Input: arr[] = [1, 23, 12, 9, 30, 2, 50], k = 3
Output: [50, 30, 23]
Explanation: Three Largest elements in the array are 50, 30 and 23.

Input: arr[] = [12, 23], k = 1
Output: [23]
Explanation: 1st Largest element in the array is 23.

Code:

```
import java.util.*;
class Main{
   static List<Integer> kLargest(int arr[], int k) {
       List<Integer> result = new ArrayList<>();
       Arrays.sort(arr);
       for (int i = arr.length - 1; i >= arr.length - k; {
          result.add(arr[i]);
       }
       return result;
   }
 public static void main(String[] args) {
       Main solution = new Main();
       int[] arr = {12, 5, 787, 1, 23};
```

```
        int k = 2;
        System.out.println(solution.kLargest(arr, k));
    }
}
```

Output: [787, 23]

Time complexity:O(nlogn)

6. Form the Largest Number

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.

Examples:

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.

Input: arr[] = [54, 546, 548, 60]
Output: "6054854654"
Explanation: Given numbers are [54, 546, 548, 60], the arrangement "6054854654" gives the largest value.

Input: arr[] = [3, 4, 6, 5, 9]
Output: "96543"
Explanation: Given numbers are [3, 4, 6, 5, 9], the arrangement "96543" gives the largest value.

Code:

```
import java.util.*;
class Solution {
    String printLargest(int[] arr) {
        String[] strArr = new String[arr.length];
        for (int i = 0; i < arr.length; i++) {
```

```
            strArr[i] = String.valueOf(arr[i]);
        }
        Arrays.sort(strArr, (a, b) -> (b + a).compareTo(a + b));
        if (strArr[0].equals("0")) {
            return "0";
        }
        StringBuilder result = new StringBuilder();
        for (String s : strArr) {
            result.append(s);
        }
        return result.toString();
    }
    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] arr = {3, 30, 34, 5, 9};
        System.out.println(solution.printLargest(arr));
    }
}
```

Output:9534330

Time Complexity:O(nlogn * m)