PRACTICE ASSIGNMENT 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]

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.util.Scanner;
public class BubbleSort {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        bubbleSort(arr, n);
        for (int num : arr) {
            System.out.print(num + " ");
        }
        sc.close();
    }
    static void bubbleSort(int[] arr, int n) {
        for (int i = 0; i < n - 1; i++) {
            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;
            }
        }
    }
  }
}
```

OUTPUT:

```
X:\Desktop\Assignments\Day6\Code_Files>javac BubbleSort.java

X:\Desktop\Assignments\Day6\Code_Files>java BubbleSort
5
4 1 3 9 7
1 3 4 7 9
```

TIME COMPLEXITY: O(n^2)

---

2. 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]

Input: arr[] = [2, 1, 6, 10, 4, 1, 3, 9, 7]
Output: [1, 1, 2, 3, 4, 6, 7, 9, 10]

Input: arr[] = [5, 5, 5, 5]
Output: [5, 5, 5, 5]

CODE:

```
import java.util.Scanner;
```

```
public class QuickSort {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        quickSort(arr, 0, n - 1);
        for (int num : arr) {
            System.out.print(num + " ");
        }
        sc.close();
    }
    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);
        }
    }
    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) {
                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:

```
X:\Desktop\Assignments\Day6\Code_Files>java QuickSort
5
4 1 3 9 7
1 3 4 7 9
```

TIME COMPLEXITY: O(nlogn)

---

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.

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:

```
import java.util.Arrays;
class Main {
    static char nonRepeatingChar(String s) {
        int[] freq = new int[26];
        Arrays.fill(freq, 0);
        for (int i = 0; i < s.length(); i++) {
            freq[s.charAt(i) - 'a']++;
        }
        for (int i = 0; i < s.length(); i++) {
```

```
            if (freq[s.charAt(i) - 'a'] == 1) {
                return s.charAt(i);
            }
        }
        return '$';
    }
    public static void main(String[] args) {
        String input = "geeksforgeeks";
        char result = nonRepeatingChar(input);
        System.out.println(result == '$' ? -1 : result);
    }
}
```

OUTPUT:

```
geeksforgeeks

Your Output:

f

Expected Output:

f
```

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:
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.

Examples:
Input: s1 = "geek", s2 = "gesek"
Output: 1

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

Input : s1 = "abc", s2 = "def"
Output: 3

CODE:

```
import java.util.Scanner;

class Solution {
    public int editDistance(String s, String t) {
        int m = s.length(), n = t.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 (s.charAt(i - 1) == t.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) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter first string: ");
        String s1 = sc.nextLine();
        System.out.print("Enter second string: ");
        String s2 = sc.nextLine();

        Solution solution = new Solution();
        int result = solution.editDistance(s1, s2);
```

```
        System.out.println("Edit Distance: " + result);

        sc.close();
    }
}
```

OUTPUT:

```
X:\Desktop\Assignments\Day6\Code_Files>java Solution
Enter first string: geek
Enter second string: gesek
Edit Distance: 1
```

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.

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

Input: arr[] = [1, 23, 12, 9, 30, 2, 50], k = 3
Output: [50, 30, 23]

CODE:

```
import java.util.*;

class LargestElements {
    public List<Integer> kLargest(int[] arr, int k) {
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        for (int num : arr) {
            pq.add(num);
            if (pq.size() > k) {
                pq.poll();
            }
        }

        List<Integer> result = new ArrayList<>(pq);
```

```
        Collections.sort(result, Collections.reverseOrder());
        return result;
    }
    public static void main(String[] args) {
        LargestElements sol = new LargestElements();
        int[] arr = {12, 5, 787, 1, 23};
        int k = 2;
        List<Integer> largestElements = sol.kLargest(arr, k);
        System.out.println(largestElements);
    }
}
```

OUTPUT:

```
X:\Desktop\Assignments\Day6\Code_Files>javac LargestElements.java

X:\Desktop\Assignments\Day6\Code_Files>java LargestElements
[787, 23]
```

TIME COMPLEXITY: O(nlogk+klogk)

---

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.

Input: arr[] = [3, 30, 34, 5, 9] Output: "9534330"
Input: arr[] = [54, 546, 548, 60] Output: "6054854654"

CODE:

```
import java.util.*;

class LargestNumber {
    public String largestNumber(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 str : strArr) {
            result.append(str);
        }
        return result.toString();
    }
    public static void main(String[] args) {
        LargestNumber sol = new LargestNumber();
        int[] arr = {3, 30, 34, 5, 9};
        String largestNum = sol.largestNumber(arr);
        System.out.println(largestNum);
    }
}
```

OUTPUT:

```
X:\Desktop\Assignments\Day6\Code_Files>javac LargestNumber.java

X:\Desktop\Assignments\Day6\Code_Files>java LargestNumber
9534330
```

TIME COMPLEXITY: O(nlogn)
