DSA Practice Solutions – 18.11.24

NAME: N J Meenakshi

DEPARTMENT: CSBS

ROLL NO.: 22CB028

1. **Bubble Sort**
   Given an array of N integers, write a program to implement the Bubble Sorting algorithm.
   Input: N = 6, array[] = {13,46,24,52,20,9}
   Output: 9,13,20,24,46,52
   Explanation: After sorting we get 9,13,20,24,46,52
   Input: N = 5, array[] = {5,4,3,2,1}
   Output: 1,2,3,4,5
   Explanation: After sorting we get 1,2,3,4,5

   Code:

```java
import java.util.Arrays;
public class Bubble_Sort {
    public static void solution(int[] arr){
    int n = arr.length;
    for (int i=0; i <n-1; i++){
        Boolean flag = 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;
            }
            flag = true;
        }
        if (flag==false) break;
    }
    System.out.println(Arrays.toString(arr));
}
    public static void main(String[] args) {
        int[] arr = {5,3,2,4};
        solution(arr);
    }
}
```

   Output: [2, 3, 4, 5]

   Time Complexity: O(N^2)

2. **Quick Sort**
   Given an array of n integers, sort the array using the **Quicksort** method.
   Input: N = 5 , Arr[] = {4,1,7,9,3}

Output: 1 3 4 7 9
Explanation: After sorting the array becomes 1, 3, 4, 7, 9
Input: N = 8 , Arr[] = {4,6,2,5,7,9,1,3}
Output: 1 2 3 4 5 6 7 9
Explanation: After sorting the array becomes 1, 3, 4, 7, 9

Code:

```
import java.util.Arrays;
public class Quick_Sort {
    public static int partition(int[] arr, int high, int low){
        int ind = arr[high];
        int i = low-1;

        for (int j=low;j<=high-1;j++){
            if (arr[j] < ind){
                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;
    }
    public static void solution(int[] arr,int low,int high){
        if (low<high){
            int ind = partition(arr, high, low);
            solution(arr,low,ind-1);
            solution(arr,ind+1,high);
        }
    }
    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        solution(arr,0,arr.length-1);
        System.out.println(Arrays.toString(arr));
    }
}
```

Output: [1, 5, 7, 8, 9, 10]

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.

Code:

```java
import java.util.Arrays;

public class Non_repeating_Char {
    public static char solution(String s){
        int[] arr = new int[26];
        Arrays.fill(arr,-1);
        for (int i = 0; i < s.length() ; i++){
            if (arr[s.charAt(i)-'a'] == -1){
                arr[s.charAt(i)-'a'] = i;
            }
            else{
                arr[s.charAt(i)-'a']= -2;
            }
        }
        int ans = Integer.MAX_VALUE;
        for (int i=0; i<26; i++){
            if (arr[i]>=0){
                ans = Math.min(ans, arr[i]);
            }
        }
        if (ans==Integer.MAX_VALUE) return '$';
        return s.charAt(ans);
    }
    public static void main(String[] args) {
        String s = "racecar";
        System.out.println(solution(s));
    }
}
```

Output: e

Time Complexity: O(N)

4. **Edit Distance**
   Given two strings s1 and s2 of lengths m and n respectively and below operations that can be performed on s1. Find the minimum number of edits (operations) to convert 's1' into 's2'.
   - **Insert:** Insert any character before or after any index of s1
   - **Remove:** Remove a character of s1
   - **Replace:** Replace a character at any index of s1 with some other character.

   Input: s1 = "geek", s2 = "gesek"
   Output: 1
   Explanation: We can convert s1 into s2 by inserting a 's' between two consecutive 'e' in s2.
   Input: s1 = "cat", s2 = "cut"
   Output: 1
   Explanation: We can convert s1 into s2 by replacing 'a' with 'u'.
   Input: s1 = "sunday", s2 = "saturday"
   Output: 3

Explanation: Last three and first characters are same. We basically need to convert “un” to “atur”. This can be done using below three operations. Replace ‘n’ with ‘r’, insert t, insert a

Code:

```
public class Edit_Distance {
    public static int solution(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;
        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][j - 1], Math.min(dp[i - 1][j], dp[i -
1][j - 1]));
            }
        }
        return dp[m][n];
    }
    public static void main(String[] args) {
        String s1 = "GEEXSFRGEEKKS";
        String s2 = "GEEKSFORGEEKS";
        System.out.println(solution(s1, s2));
        }
}
```

Output: 3

Time Complexity: O(M*N)

5. **K-th Largest Element**

Given an array arr[] of N distinct elements and a number K, where K is smaller than the size of the array. Find the K’th smallest element in the given array.

Input: arr[] = {7, 10, 4, 3, 20, 15}, K = 3

Output: 7

Input: arr[] = {7, 10, 4, 3, 20, 15}, K = 4

Output: 10

Code:

```
import java.util.Arrays;
public class Kth_Largest_Element {
    static int solution(int[] arr, int n, int k)
    {
        int m = arr[0];
        for (int i = 1; i < n; i++) {
            if (arr[i] > m) {
```

```
                m = arr[i];
            }
        }
        int[] freq = new int[m + 1];
        Arrays.fill(freq, 0);
        for (int i = 0; i < n; i++) {
            freq[arr[i]]++;
        }
        int c = 0;
        for (int i = 0; i <= m; i++) {
            if (freq[i] != 0) {
                c += freq[i];
                if (c >= k) {
                    return i;
                }
            }
        }
        return -1;
    }
    public static void main(String[] args)
    {
        int[] arr = {12,5,2,9,6,3,7};
        int n = arr.length;
        int k = 2;
        System.out.println(solution(arr, n, k));
    }
}
```

Output: 3

Time Complexity: O(N+M)

6. **Form the Largest Number**

Given an array of strings arr[] of length n, where every string representing a non-negative integers, the task is to arrange them in a manner such that after concatenating them in order, it results in the largest possible number. Since the result may be very large, return it as a string.

Input: n = 5, 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: n = 4, arr[] = {“54”, “546”, “548”, “60”}

Output: “6054854654”

Explanation: Given numbers are {“54”, “546”, “548”, “60”}, the arrangement “6054854654” gives the largest value.

Code:

```
import java.util.Arrays;
import java.util.Comparator;
public class Form_largest_Number {
    public static String solution(String[] arr){
        Comparator<String> comp = (x,y) -> (x+y).compareTo(y+x);
        Arrays.sort(arr, comp.reversed());
        if (arr[0].equals("0")) {
```

```
            return "0";
        }
        StringBuilder result = new StringBuilder();
        for (String num : arr) {
            result.append(num);
        }
        return result.toString();
    }
    public static void main(String[] args)
    {
        String[] arr = {"3","30","34","5","9"};
        System.out.println(solution(arr));
    }
}
```

Output: 9534330

Time Complexity: O(NlogN)