18/11/2024 DSA Problems

1.Bubble Sort

Solution:

```
import java.util.*;
class BubbleSort{
    public static void bubbleSort(int arr[]) {
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }
    public static void main(String[] args) {
        int[] arr = {3,5,2,8,6};
            bubbleSort(arr);
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

Output:

```
D:\OOPS\18-11-2024 practice.java>javac BubbleSort.java

D:\OOPS\18-11-2024 practice.java>java BubbleSort
2 3 5 6 8
```

Time Complexity: O(n^2)

Space Complexity:O(1)

2. Quick Sort

Solution:

```
import java.util.*;
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 + " ");
        }
    }
}
```

Output:

```
D:\OOPS\18-11-2024 practice.java>javac QuickSort.java

D:\OOPS\18-11-2024 practice.java>java QuickSort
1 5 7 8 9 10
```

Time Complexity: O(n^2)

Space Complexity: O(n)

3. **Find non-repeating character of given string**

Given a string **s** of **lowercase** English letters, the task is to find the **first non-repeating** character. If there is no such character, return **'$'**.

Examples:

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

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

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

Solution:

```
import java.util.*;

class NonRepeatChar {

    static final int MAX_CHAR = 26;

    static char nonRepeatingChar(String s) {

        int[] freq = new int[MAX_CHAR];

        for (char c : s.toCharArray())

            freq[c - '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 s = "racecar";

        System.out.println(nonRepeatingChar(s));

    }
```

}

Output:

```
D:\OOPS\18-11-2024 practice.java>javac NonRepeatChar.java

D:\OOPS\18-11-2024 practice.java>java NonRepeatChar
e
```

Time Complexity: O(n)

Space Complexity: O(1)

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.

Note: All of the above operations are of equal cost.

Examples:

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*

Solution:

Output:

```
D:\OOPS\18-11-2024 practice.java>javac EditDistance.java

D:\OOPS\18-11-2024 practice.java>java EditDistance
3
```

```
import java.util.*;

public class EditDistance {

    public static int editDist(String s1, String s2) {

        int m = s1.length(), n = s2.length();

        int[] prev = new int[n + 1], curr = new int[n + 1];
```

```
        for (int j = 0; j <= n; j++) prev[j] = j;
        for (int i = 1; i <= m; i++) {
            curr[0] = i;
            for (int j = 1; j <= n; j++) {
                if (s1.charAt(i - 1) == s2.charAt(j - 1)) curr[j] = prev[j - 1];
                else curr[j] = 1 + Math.min(curr[j - 1], Math.min(prev[j], prev[j - 1]));
            }
            int[] temp = prev;
            prev = curr;
            curr = temp;
        }
        return prev[n];
    }

    public static void main(String[] args) {
        String s1 = "GEEXSFRGEEKKS", s2 = "GEEKSFORGEEKS";
        System.out.println(editDist(s1, s2));
    }
}
```

Time Complexity: O(m*n)

Space Complexity: O(n)

5. **Find k largest elements in an array**

Given an array **arr[]** and an integer **k**, the task is to find **k largest** elements in the given array. Elements in the output array should be in decreasing order.

Examples:

Input: *[1, 23, 12, 9, 30, 2, 50], K = 3*
Output: [*50, 30, 23]*

Input: *[11, 5, 12, 9, 44, 17, 2], K = 2*
Output: [*44, 17]*

Solution:

```
import java.util.*;
class KLargestElement {
    static ArrayList<Integer> kLargest(int[] arr, int k) {
        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;
    }

    public static void main(String[] args) {
        int[] arr = {1, 23, 12, 9, 30, 2, 50};
        int k = 3;
        ArrayList<Integer> res = kLargest(arr, k);
        for (int ele : res) System.out.print(ele + " ");
    }
}
```

Output:

```
D:\OOPS\18-11-2024 practice.java>javac KLargestElement.java

D:\OOPS\18-11-2024 practice.java>java KLargestElement
50 30 23
```

Time Complexity: O(n*log(n))

Space Complexity: O(1)

6. **Find the largest Number that can be formed with the given Digits**

Given an array of integers arr[] represents digits of a number. The task is to write a program to generate the largest number possible using these digits.

Note: The digits in the array are between 0 and 9. That is, 0 < arr[i] < 9.

Examples:

Input: arr[] = {4, 7, 9, 2, 3}
Output: Largest number: 97432

Input: arr[] = {8, 6, 0, 4, 6, 4, 2, 7}
Output: Largest number: 87664420

Solution:

```
import java.util.*;

class FindMaxNum {

    static void findMaxNum(int arr[], int n) {

        int[] hash = new int[10];

        for (int i = 0; i < n; i++) hash[arr[i]]++;

        for (int i = 9; i >= 0; i--)
```

```
        for (int j = 0; j < hash[i]; j++)
            System.out.print(i);
    }
    public static void main(String[] args) {
        int arr[] = { 1, 2, 3, 4, 5, 0 };
        int n = arr.length;
        findMaxNum(arr, n);
    }
}
```

Output:

```
D:\OOPS\18-11-2024 practice.java>javac FindMaxNum.java

D:\OOPS\18-11-2024 practice.java>java FindMaxNum
543210
```

Time Complexity: O(n)

Space Complexity: O(1)