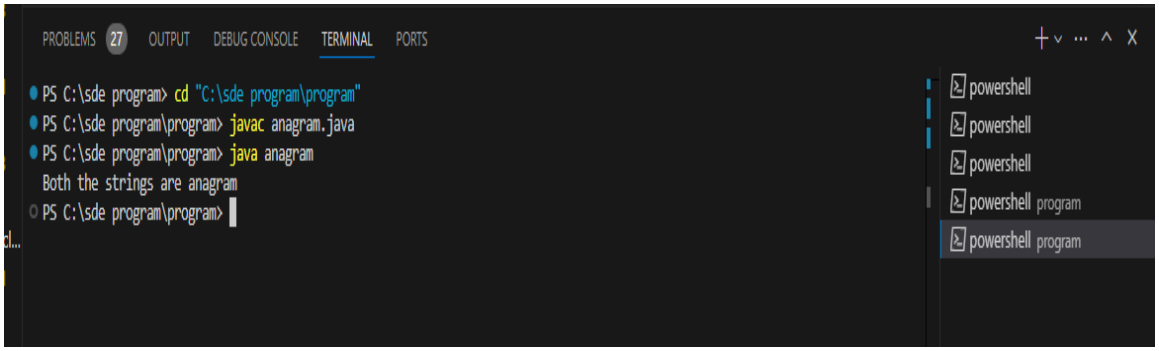
Coding Practice Problems(12/11/2024)

Q 1) Anagram Program :

```
import java.util.Arrays;

public class anagram {

   public static void main(String[] args){

      String str1="Brag";

      String str2="Grab";

      str1=str1.toLowerCase();

      str2=str2.toLowerCase();

      if(str1.length()!=str2.length()){

         System.out.println("Both the string are not anagram");

      }

      else{

         char[]string1=str1.toCharArray();

         char[]string2=str2.toCharArray();

         Arrays.sort(string1);

         Arrays.sort(string2);

         if (Arrays.equals(string1,string2)==true){

            System.out.println("Both the strings are anagram");

         }

         else{

            System.out.println("Both the strings are not anagram");

         }

      }

   }

}
```

Time Complexity: O(n)

Output:



Q 2) Row with max 1s'

```
import java.util.*;

class maximum_rows {

static int R = 4 ;

static int C = 4 ;

static int rowWithMax1s(int mat[][], int R, int C)

{

   boolean flag = true;

   int max_row_index = 0, max_ones = 0;;

   for(int i = 0 ; i < R ; i++){

         int count1 = 0 ;

         for(int j = 0 ; j < C ; j++){

            if(mat[i][j] == 1){

               count1++;

               flag = false;

            }

         }

         if(count1>max_ones){

            max_ones = count1;
```

```
            max_row_index = i;
        }
    }
  if(flag){
        return -1;
    }
  return max_row_index;
}
  public static void main(String[] args) {
  int mat[][] = { {0, 0, 0, 1},
                {0, 1, 1, 1},
                {1, 1, 1, 1},
                {0, 0, 0, 0}};
  System.out.print("Index of row with maximum 1s is " + rowWithMax1s(mat,R,C));
  }
}
```

Time Complexity: O(m*n)

Output:

Q 3) Longest consecutive subsequence

```
import java.io.*;
import java.util.*;
class longest_consequence {
  static int findLongestConseqSubseq(int arr[], int n)
```

```
    {
        Arrays.sort(arr);
        int ans = 0, count = 0;
        ArrayList<Integer> v = new ArrayList<Integer>();
        v.add(10);
        for (int i = 1; i < n; i++) {
            if (arr[i] != arr[i - 1])
                v.add(arr[i]);
        }
        for (int i = 0; i < v.size(); i++) {
            if (i > 0 && v.get(i) == v.get(i - 1) + 1)
                count++;
            else
                count = 1;
            ans = Math.max(ans, count);
        }
        return ans;
    }
    public static void main(String[] args)
    {
        int arr[] = { 1, 9, 3, 10, 4, 20, 2 };
        int n = arr.length;
        System.out.println(
            "Length of the Longest "+ "contiguous subsequence is "+ findLongestConseqSubseq(arr, n));
    }
}
```

Time Complexity: O(Nlog(N)

Output:

Q 4) Longest Palindrome in a String

```
public class longest_palindrome {
    static boolean checkPal(String s, int low, int high) {
        while (low < high) {
            if (s.charAt(low) != s.charAt(high))
                return false;
            low++;
            high--;
        }
        return true;
    }
    static String longestPalSubstr(String s) {
        int n = s.length();
        int maxLen = 1, start = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i; j < n; j++) {
                if (checkPal(s, i, j) && (j - i + 1) > maxLen) {
                    start = i;
                    maxLen = j - i + 1;
                }
```

```
        }
      }
      return s.substring(start, start + maxLen);
    }
    public static void main(String[] args) {
      String s = "forgeeksskeegfor";
      System.out.println(longestPalSubstr(s));
    }
}
```

Time Complexity: O(N3)

Output:

Q 5) Rat in a Maze Problem

```
import java.util.ArrayList;
import java.util.List;
public class race_in_maze {
    static String direction = "DLRU";
    static int[] dr = { 1, 0, 0, -1 };
    static int[] dc = { 0, -1, 1, 0 };
    static boolean isValid(int row, int col, int n,int[][] maze)
    {
      return row >= 0 && col >= 0 && row < n && col < n && maze[row][col] == 1;
    }
```

```
    static void findPath(int row, int col, int[][] maze,int n, ArrayList<String> ans,StringBuilder
currentPath)
    {
        if (row == n - 1 && col == n - 1) {
            ans.add(currentPath.toString());
            return;
        }
        maze[row][col] = 0;
        for (int i = 0; i < 4; i++) {
            int nextrow = row + dr[i];
            int nextcol = col + dc[i];
            if (isValid(nextrow, nextcol, n, maze)) {
                currentPath.append(direction.charAt(i));
                findPath(nextrow, nextcol, maze, n, ans,currentPath);
                currentPath.deleteCharAt(currentPath.length() - 1);
            }
        }
        maze[row][col] = 1;
    }

    public static void main(String[] args)
    {
        int[][] maze = { { 1, 0, 0, 0 },
                         { 1, 1, 0, 1 },
                         { 1, 1, 0, 0 },
                         { 0, 1, 1, 1 } };

        int n = maze.length;
```

```
        ArrayList<String> result = new ArrayList<>();
        StringBuilder currentPath = new StringBuilder();
        if (maze[0][0] != 0 && maze[n - 1][n - 1] != 0) {
            findPath(0, 0, maze, n, result, currentPath);
        }
        if (result.size() == 0)
            System.out.println(-1);
        else
            for (String path : result)
                System.out.print(path + " ");
        System.out.println();
    }
}
```

Time Complexity: O(n^3)

Output: