Analysis and Design of Algorithms (3150703)

Open Ended Problem

Group Members :

- Bagrecha Shagun (180060107003)
- Bajaj Vibhuti (180060107004)
- Ashish Patel (180060107036)
- Patel Dhruv (180060107038)

Problem Statement

"From the given String, find maximum size possible palindromic sequence."

What is a Palindrome?

A Palindrome is a word, number, phrase, or other sequence of characters which reads the same backward as forward.

Examples of palindromic words are ***MADAM, RACECAR***, etc.

Also, there exists some palindromic phrases as well such as ***Never odd or even.***

Aim

In this problem we will be given an input string. The primary aim of our algorithm will be to somehow compute the length of Maximum subsequence of this string which is palindromic and later report the Palindromic String itself.

Strategy

This problem is somewhat similar to LCS (Longest Common Subsequence) problem except that we do not have a second string to check the LCS.

We can solve this problem using the same logic of we make some clever modifications. *What if we derive a second String from the input string, which is simply the reverse of the input string, and find the LCS?*

This indeed solves the problem.

Since our problem follows the principle of optimality, our algorithmic strategy becomes **Dynamic Programming**. In fact, it is a slight variation of LCS itself!

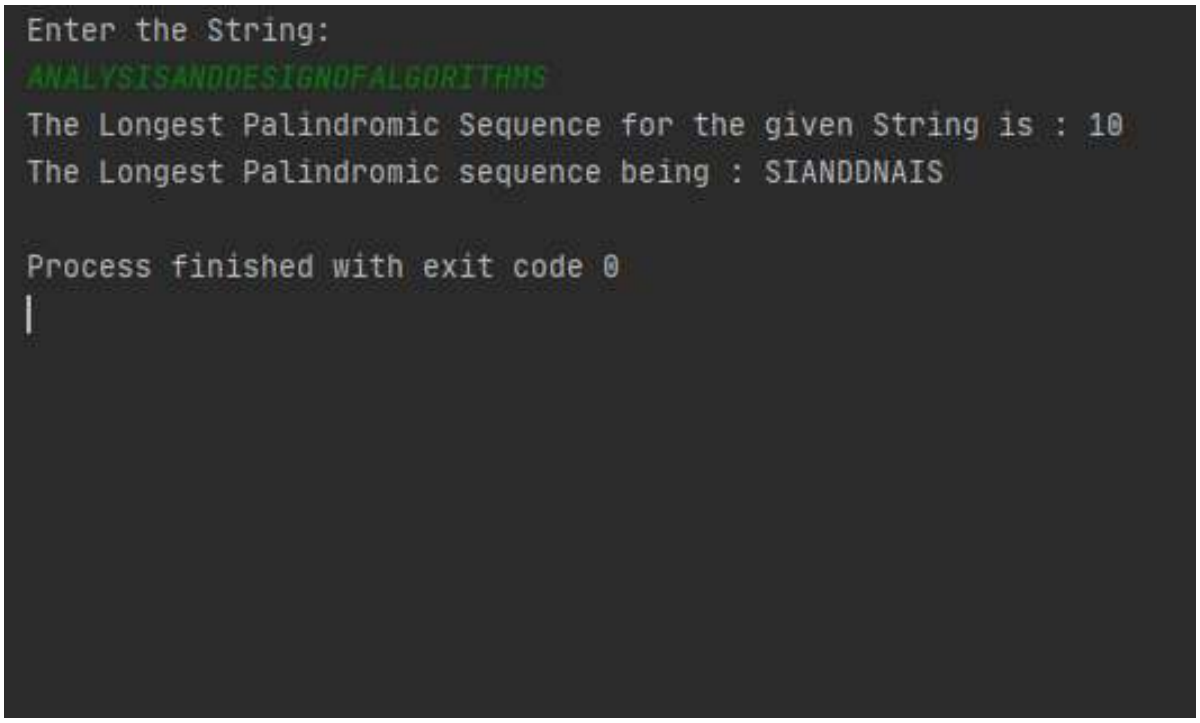
Code

```
import java.util.Scanner;
public class OEP_LPS {
    private static int L[][];
    private static char[] a1;
    private static char[] a2;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the String: ");
        String s1 = sc.nextLine();
        String s2 = "";
```

```
        for(int i=s1.length()-1;i>=0;i--) {
            s2 += s1.charAt(i);
        }

        a1 = s1.toCharArray();
        a2 = s2.toCharArray();

        System.out.println("The Longest Palindromic Sequence for the
given String is : "+lcs(a1,a2,s1.length()));
        printLPS(a1.length);
    }
    private static int lcs( char[] X, char[] Y,int n ) {
        L = new int[n+1][n+1];
        for (int i=0; i<=n; i++) {
            for (int j=0; j<=n; j++) {
                if (i == 0 || j == 0) {
                    L[i][j] = 0;
                }else if (X[i-1] == Y[j-1]) {
                    L[i][j] = L[i-1][j-1] + 1;
                }else {
                    L[i][j] = Math.max(L[i-1][j], L[i][j-1]);
                }
            }
        }
        return L[n][n];
    }

// Method to print our result.
    private static void printLPS(int n) {
        int index = L[n][n];
```

```
        char[] lcs = new char[index + 1];

        int i = n, j = n;
        while (i > 0 && j > 0) {
            if (a1[i - 1] == a2[j - 1]) {
                lcs[index - 1] = a1[i - 1];
                i--;
                j--;
                index--;
            } else if (L[i - 1][j] > L[i][j - 1]) {
                i--;
            } else {
                j--;
            }
        }
        String ans = "";
        for (int x = 0; x < lcs.length; x++) {
            ans += lcs[x];
        }
        System.out.println("The Longest Palindromic sequence being :
" + ans);
    }
}
```

Output



Time Complexity

This algorithm takes **$O(n^2)$** time.

The code gives a clear picture for this. We can clearly observe the filling of table takes quadratic time.