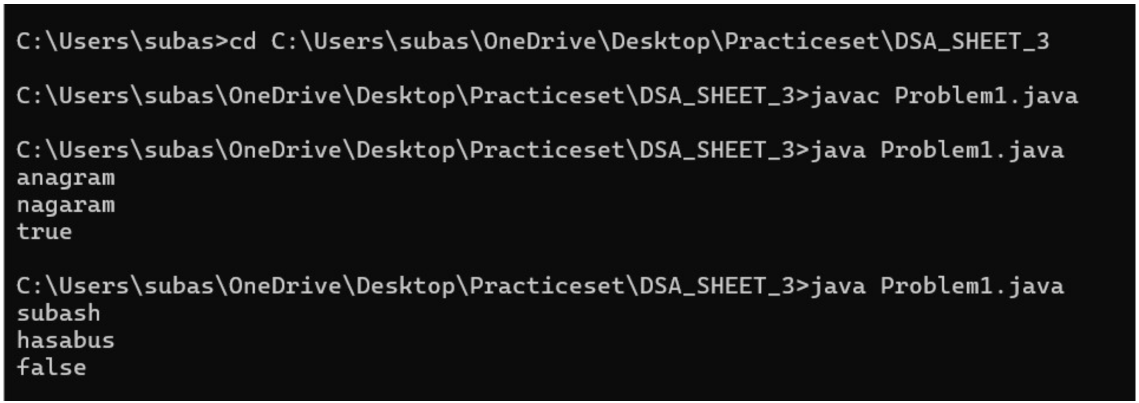
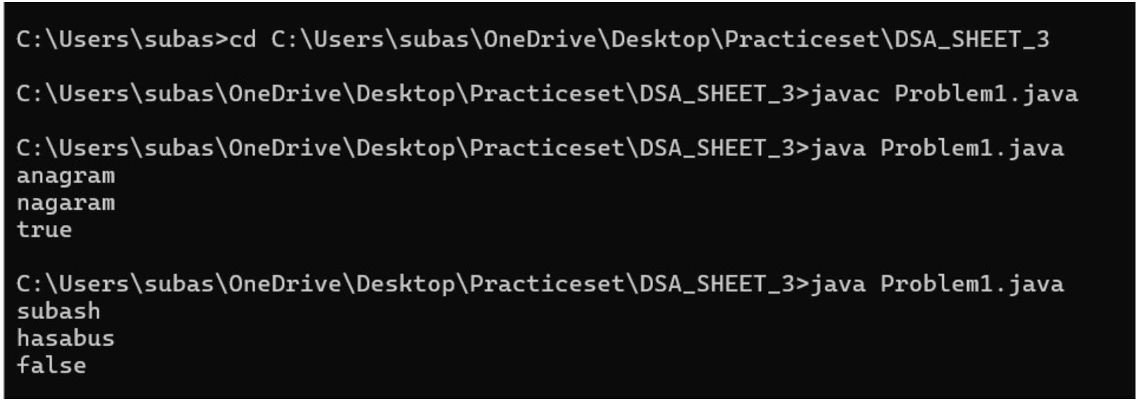
DSA PRACTICE 3

1.Given two strings s and t, return true if t is an anagramof s, t and false otherwise.

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

class Problem1 {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String s = sc.nextLine();
    String t = sc.nextLine();
    System.out.println(isAnagram(s, t));
    sc.close();
  }

  public static boolean isAnagram(String s, String t) {
    if(s.length()!= t.length()) return false;
    HashMap<Character, Integer> map1= new HashMap<>();
    HashMap<Character, Integer> map2= new HashMap<>();
     for(int i=0; i<s.length();i++){
       map1.put(s.charAt(i), map1.getOrDefault(s.charAt(i),0)+1);
    }
    for(int i=0; i<t.length();i++){
      map2.put(t.charAt(i), map2.getOrDefault(t.charAt(i),0)+1);
    }

    for(char k: map1.keySet()){
      if(!map1.get(k).equals(map2.get(k))) return false;
    }
    return true;
    }
}
```



```
C:\Users\subas>cd C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>javac Problem1.java

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>java Problem1.java
anagram
nagaram
true

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>java Problem1.java
subash
hasabus
false
```



```
C:\Users\subas>cd C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>javac Problem1.java

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>java Problem1.java
anagram
nagaram
true

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>java Problem1.java
subash
hasabus
false
```

Time Complexity : O(N)

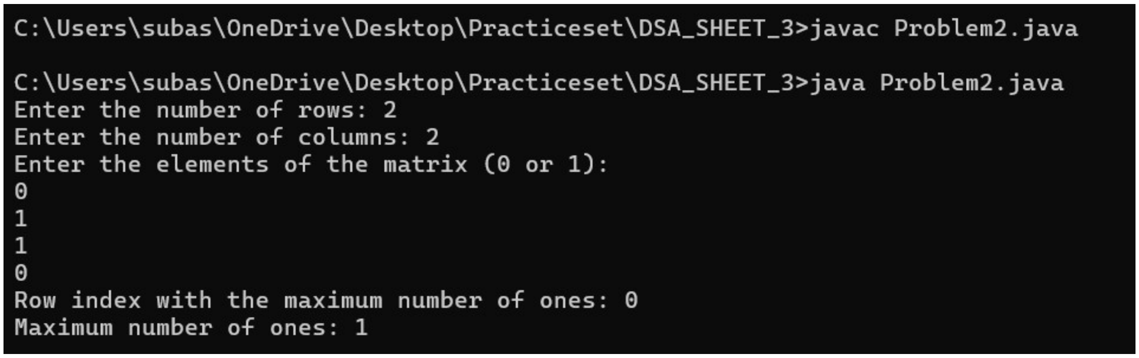
Space Complexity: O(N)

2. Given a m x n binary matrix mat, find the 0-indexed position of the row that contains the maximum count of ones, and the number of ones in that row.

In case there are multiple rows that have the maximum count of ones, the row with the smallest row number should be selected.

Return *an array containing the index of the row, and the number of ones in it.*

```
import java.util.Scanner;


public class Problem2 {

    public int[] rowAndMaximumOnes(int[][] mat) {

        int max = 0, index = 0;

        for (int i = 0; i < mat.length; i++) {
            int count = 0;
            for (int j = 0; j < mat[i].length; j++) {
                count += mat[i][j];
            }
            if (count > max) {
                max = count;
                index = i;
            }
        }
        return new int[]{index, max};
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of rows: ");
        int rows = scanner.nextInt();
        System.out.print("Enter the number of columns: ");
        int cols = scanner.nextInt();

        int[][] mat = new int[rows][cols];

        System.out.println("Enter the elements of the matrix (0 or 1):");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                mat[i][j] = scanner.nextInt();
            }
        }

        scanner.close();
        Problem2 solution = new Problem2();

        int[] result = solution.rowAndMaximumOnes(mat);

        System.out.println("Row index with the maximum number of ones: " + result[0]);

        System.out.println("Maximum number of ones: " + result[1]);

    }

}
```



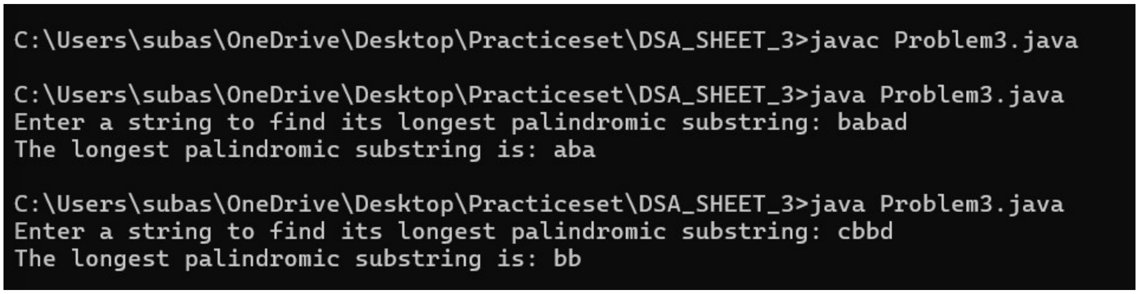
```
C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>javac Problem2.java

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>java Problem2.java
Enter the number of rows: 2
Enter the number of columns: 2
Enter the elements of the matrix (0 or 1):
0
1
1
0
Row index with the maximum number of ones: 0
Maximum number of ones: 1
```

Time Complexity : O(N)

Space Complexity: O(1)

3. Given a string s, return *the longest palindromic substring* in s.

```
import java.util.Scanner;


public class Solution {

    public String longestPalindrome(String s) {

        if (s == null || s.length() == 0) {

            return "";

        }


        int start = 0;

        int end = 0;


        for (int i = 0; i < s.length(); i++) {

            int odd = expandAroundCenter(s, i, i);

            int even = expandAroundCenter(s, i, i + 1);
            int max_len = Math.max(odd, even);

            if (max_len > end - start) {
                start = i - (max_len - 1) / 2;
                end = i + max_len / 2;
            }
        }
        return s.substring(start, end + 1);
    }

    private int expandAroundCenter(String s, int left, int right) {
        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
            left--;
            right++;
        }
        return right - left - 1;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter a string
        System.out.print("Enter a string to find its longest palindromic substring: ");
        String input = scanner.nextLine();

        // Create an instance of Solution and call longestPalindrome
        Solution solution = new Solution();
        String longestPalindrome = solution.longestPalindrome(input);

        // Display the result
        System.out.println("The longest palindromic substring is: " + longestPalindrome);
        // Close the scanner

        scanner.close();

    }

  }
```



```
C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>javac Problem3.java

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>java Problem3.java
Enter a string to find its longest palindromic substring: babad
The longest palindromic substring is: aba

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>java Problem3.java
Enter a string to find its longest palindromic substring: cbbd
The longest palindromic substring is: bb
```

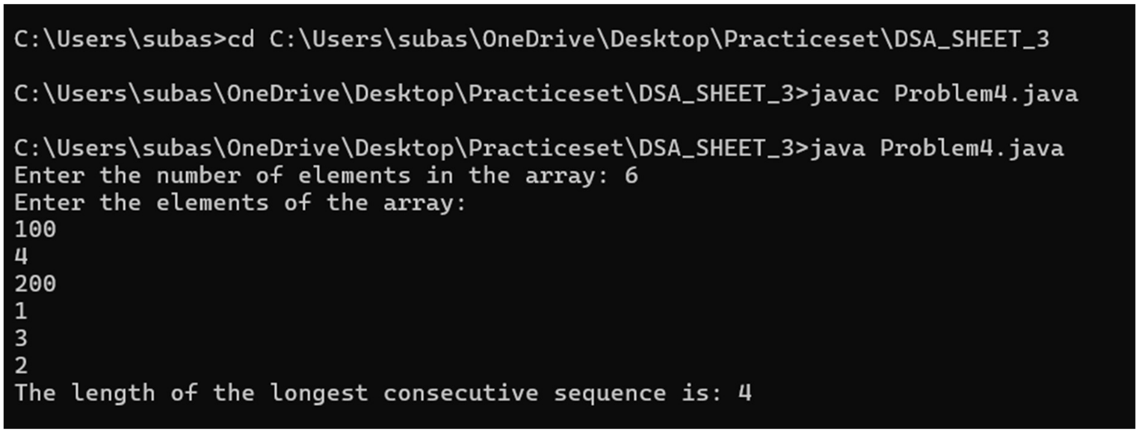
Time Complexity : O(N)

Space Complexity: O(1)

4. Given an unsorted array of integers nums, return *the length of the longest consecutive elements sequence.*

You must write an algorithm that runs in O(n) time.

```
class Solution {

  public int longestConsecutive(int[] nums) {


   if (nums.length == 1) {

      return 1;

    }

   int max=0;

    HashSet<Integer> a = new HashSet<>();


    for (int n: nums) {

      a.add(n);

    }

    for (int num : a) {

      if (!a.contains(num -1)) {
       int h=num;

       int c=1;


      while(a.contains(h+1))

      {

        h++;

        c++;

      }

      max=Math.max(max,c);

      }

    }

    return max;

  }

}
```



```
C:\Users\subas>cd C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>javac Problem4.java

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>java Problem4.java
Enter the number of elements in the array: 6
Enter the elements of the array:
100
4
200
1
3
2
The length of the longest consecutive sequence is: 4
```

Time Complexity : O(N)

Space Complexity: O(N)

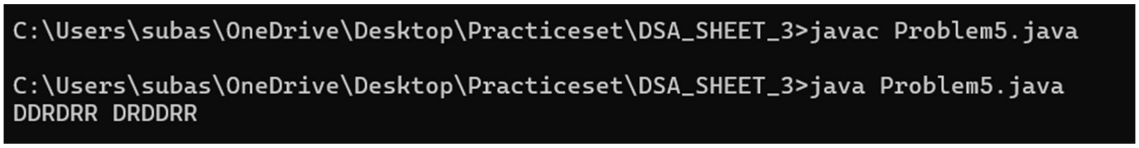
5. We have discussed Backtracking and Knight's tour problem in Set 1. Let us discuss Rat in a Maze as another example problem that can be solved using Backtracking.

Consider a rat placed at **(0, 0)** in a square matrix of order **N * N**. It has to reach the destination at **(N – 1, N – 1)**. Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are **'U'(up)**, **'D'(down)**, **'L' (left)**, **'R' (right)**. Value **0** at a cell in the matrix represents that it is blocked and rat cannot move to it while value **1** at a cell in the matrix represents that rat can be travel through it. Return the list of paths in lexicographically increasing order.

Note: In a path, no cell can be visited more than one time. If the source cell is **0**, the rat cannot move to any other cell.

```
import java.util.ArrayList;
import java.util.List;


public class MazePaths {
    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<>();
        // Store current path
        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();
    }
```

```
}
```



```
C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>javac Problem5.java

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>java Problem5.java
DDRDRR DRDDRR
```

Time Complexity : O(m*n)

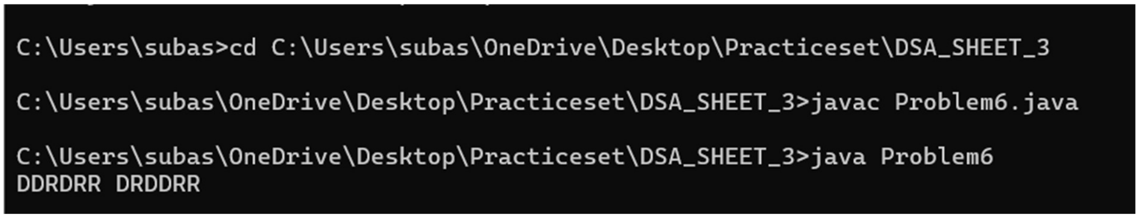
Space complexity : O(N)

6.Rat and the Maze:

```
import java.util.ArrayList;


public class Problem6 {
    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();
    }
}
```



```
C:\Users\subas>cd C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>javac Problem6.java

C:\Users\subas\OneDrive\Desktop\Practiceset\DSA_SHEET_3>java Problem6
DDRDRR DRDDRR
```

Time Complexity : O(m*n)

Space complexity : O(N)