1)anagram program

Code:

```
import java.util.Arrays;
import java.util.Scanner;

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

        System.out.print("Enter the first string: ");
        String first = scanner.nextLine();

        System.out.print("Enter the second string: ");
        String second = scanner.nextLine();

        if (isAnagram(first, second)) {
            System.out.println("The strings are anagrams.");
        } else {
            System.out.println("The strings are not anagrams.");
        }
    }

    public static boolean isAnagram(String str1, String str2) {
        char[] arr1 = str1.replaceAll("\\s", "").toLowerCase().toCharArray();
        char[] arr2 = str2.replaceAll("\\s", "").toLowerCase().toCharArray();
        Arrays.sort(arr1);
        Arrays.sort(arr2);
        return Arrays.equals(arr1, arr2);
    }
}
```

Output:

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\OneDrive\Desktop\c3>javac AnagramCheck.java

C:\Users\ASUS\OneDrive\Desktop\c3>java AnagramCheck
Enter the first string: hello
Enter the second string: world
The strings are not anagrams.

C:\Users\ASUS\OneDrive\Desktop\c3>
```

Time complexity:O(n logn)

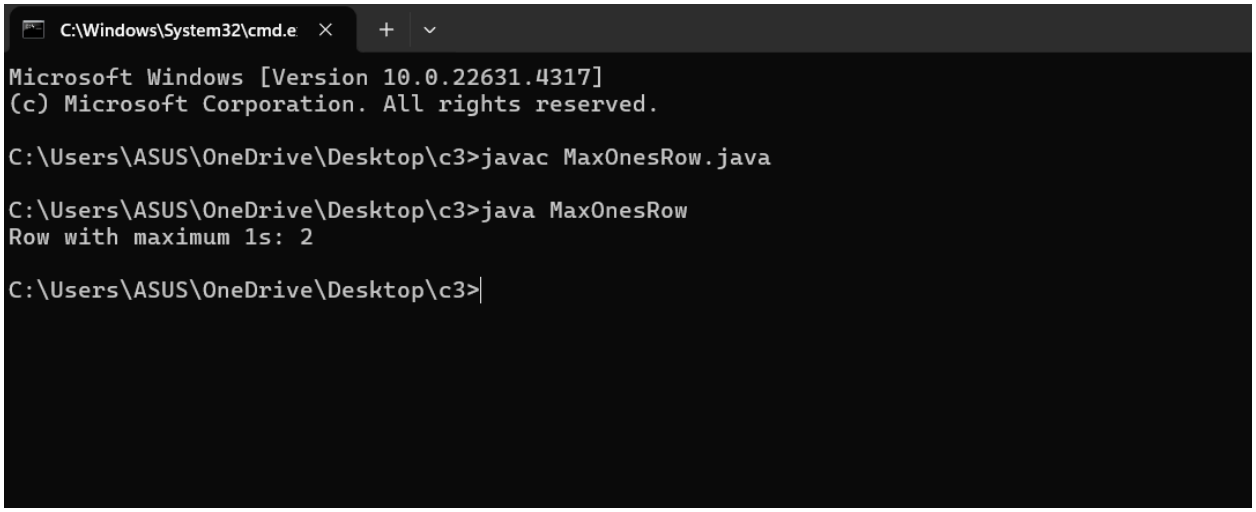
2)row with max 1s'

Code:

```
public class MaxOnesRow {
    public static void main(String[] args) {
        int[][] matrix = {
            {0, 0, 0, 1},
            {0, 1, 1, 1},
            {1, 1, 1, 1},
            {0, 0, 0, 0}
        };

        int rowIndex = rowWithMaxOnes(matrix);
        System.out.println("Row with maximum 1s: " + rowIndex);
    }

    public static int rowWithMaxOnes(int[][] matrix) {
        int maxRow = -1;
        int maxOnes = 0;
```

```
        for (int i = 0; i < matrix.length; i++) {
            int count = countOnes(matrix[i]);
            if (count > maxOnes) {
                maxOnes = count;
                maxRow = i;
            }
        }

        return maxRow;
    }

    public static int countOnes(int[] row) {
        int low = 0, high = row.length - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (row[mid] == 1) {
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }
        return row.length - low;
    }
}
```

Output:



Time complexity:O(n log m)

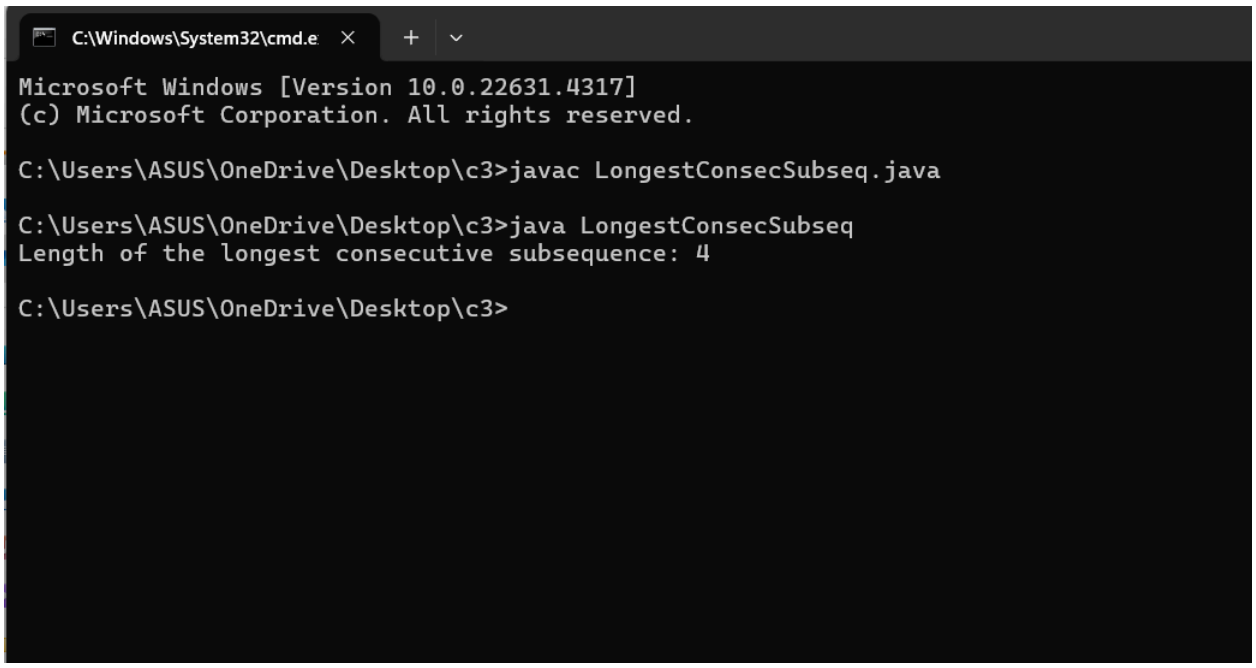
3)Longest consequtive subsequence

Code:

```
import java.util.HashSet;

public class LongestConsecutiveSubsequence {
    public static void main(String[] args) {
        int[] arr = {100, 4, 200, 1, 3, 2};
        System.out.println("Length of the longest consecutive subsequence: "
+ findLongestConsecutiveSubsequence(arr));
    }

    public static int findLongestConsecutiveSubsequence(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
        for (int num : nums) {
            set.add(num);
        }

        int longestStreak = 0;

        for (int num : nums) {
            if (!set.contains(num - 1)) {
```

```
            int currentNum = num;
            int currentStreak = 1;

            while (set.contains(currentNum + 1)) {
                currentNum++;
                currentStreak++;
            }

            longestStreak = Math.max(longestStreak, currentStreak);
        }
    }

    return longestStreak;
  }
}
```

Output:



Time complexity:O(n)

4)longest palindrome in a string

Code:

```
public class LongestPalindrome {
    public static void main(String[] args) {
        String str = "babad";
        System.out.println("Longest Palindromic Substring: " +
longestPalindrome(str));
    }

    public static String longestPalindrome(String s) {
        if (s == null || s.length() < 1) return "";

        int start = 0, end = 0;

        for (int i = 0; i < s.length(); i++) {
            int len1 = expandAroundCenter(s, i, i);
            int len2 = expandAroundCenter(s, i, i + 1);
            int len = Math.max(len1, len2);

            if (len > end - start) {
                start = i - (len - 1) / 2;
                end = i + len / 2;
            }
        }

        return s.substring(start, end + 1);
    }

    private static 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;
    }
}
```

Output:

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\OneDrive\Desktop\c3>javac  LongestPalindrome.java

C:\Users\ASUS\OneDrive\Desktop\c3>java  LongestPalindrome
Longest Palindromic Substring: aba

C:\Users\ASUS\OneDrive\Desktop\c3>
```

Time complexity:O(n^2)

5)rat in a maze problem

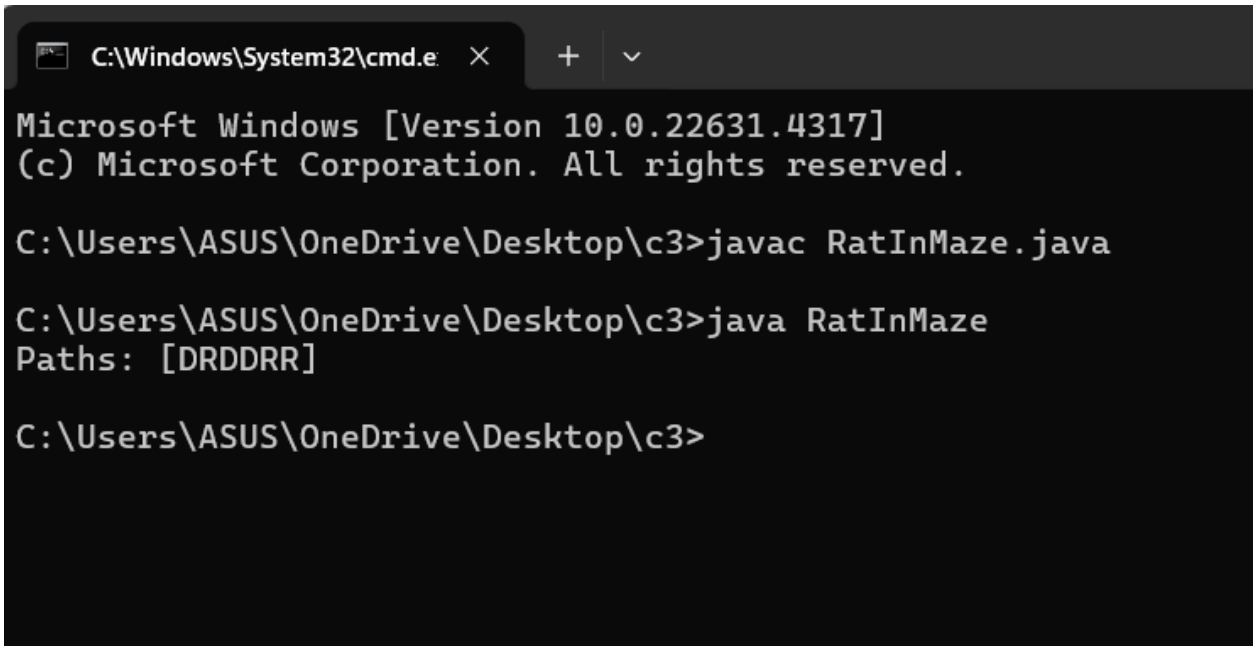
Code:

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

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

        List<String> paths = findPaths(maze);
        if (paths.isEmpty()) {
```

```
            System.out.println("No path found.");
        } else {
            System.out.println("Paths: " + paths);
        }
    }

    public static List<String> findPaths(int[][] maze) {
        List<String> paths = new ArrayList<>();
        if (maze[0][0] == 0 || maze[maze.length - 1][maze[0].length - 1] == 0) {
            return paths;
        }
        boolean[][] visited = new boolean[maze.length][maze[0].length];
        solve(maze, 0, 0, "", visited, paths);
        return paths;
    }

    private static void solve(int[][] maze, int x, int y, String path, boolean[][]
visited, List<String> paths) {
        if (x == maze.length - 1 && y == maze[0].length - 1) {
            paths.add(path);
            return;
        }

        if (!isSafe(maze, x, y, visited)) {
            return;
        }

        visited[x][y] = true;

        solve(maze, x + 1, y, path + "D", visited, paths);
        solve(maze, x, y - 1, path + "L", visited, paths);
        solve(maze, x, y + 1, path + "R", visited, paths);
        solve(maze, x - 1, y, path + "U", visited, paths);

        visited[x][y] = false;
```

```
    }

    private static boolean isSafe(int[][] maze, int x, int y, boolean[][] visited) {
        return x >= 0 && x < maze.length && y >= 0 && y < maze[0].length &&
maze[x][y] == 1 && !visited[x][y];
    }
}
```

Output:



Time complexity:O(4^(n*m))