

1. Anagram Program

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

```
class Anagram {  
    static boolean areAnagram(String str1, String str2) {  
        if (str1.length() != str2.length()) {  
            return false;  
        }  
  
        char[] arr1 = str1.toCharArray();  
        char[] arr2 = str2.toCharArray();  
  
        Arrays.sort(arr1);  
        Arrays.sort(arr2);  
  
        return Arrays.equals(arr1, arr2);  
    }  
  
    public static void main(String[] args) {  
        String str1 = "listen";  
        String str2 = "silent";  
  
        if (areAnagram(str1, str2)) {  
            System.out.println("Yes, the strings are anagrams.");  
        } else {  
            System.out.println("No, the strings are not anagrams.");  
        }  
    }  
}
```

Output

Yes, the strings are anagrams.

=== Code Execution Successful ===

Time Complexity: $O(n \log n)$

Space Complexity: $O(n)$

2. Row with Maximum 1s

```
class Max1sRow {  
    static int rowWithMax1s(int[][] mat, int n, int m) {  
        int max_row_index = -1;  
        int max_1s = -1;  
  
        for (int i = 0; i < n; i++) {  
            int count = 0;  
            for (int j = 0; j < m; j++) {  
                if (mat[i][j] == 1) {  
                    count++;  
                }  
            }  
            if (count > max_1s) {  
                max_1s = count;  
                max_row_index = i;  
            }  
        }  
        return max_row_index;  
    }  
}
```

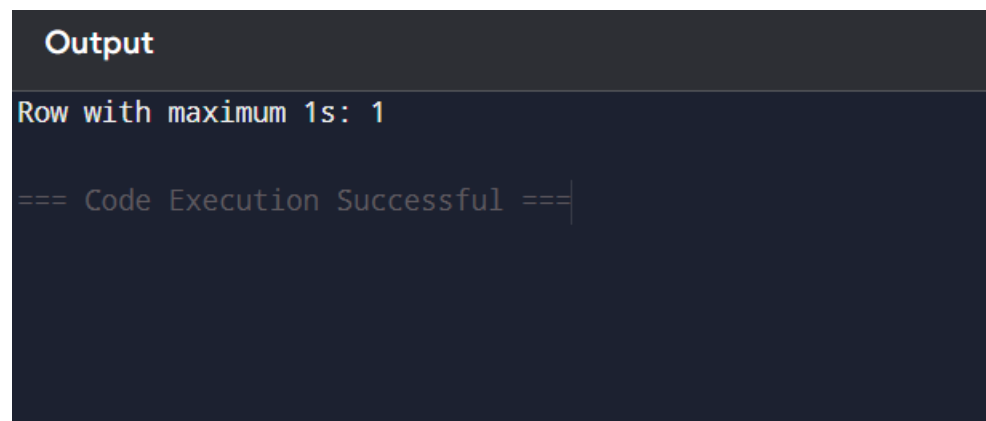
```

    }

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

        System.out.println("Row with maximum 1s: " + rowWithMax1s(mat, 4, 4));
    }
}

```



```

Output
Row with maximum 1s: 1
=== Code Execution Successful ===

```

Time Complexity: $O(n * m)$

Space Complexity: $O(1)$

3. Longest Consecutive Subsequence

```

import java.util.*;

class LongestConsecutiveSubsequence {
    static int findLongestConseqSubseq(int[] arr, int n) {
        Set<Integer> s = new HashSet<>();
        for (int num : arr) {
            s.add(num);

```

```
}
```

```
int longestStreak = 0;
```

```
for (int num : arr) {
```

```
    if (!s.contains(num - 1)) {
```

```
        int currentNum = num;
```

```
        int currentStreak = 1;
```

```
        while (s.contains(currentNum + 1)) {
```

```
            currentNum++;
```

```
            currentStreak++;
```

```
        }
```

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

```
    }
```

```
}
```

```
return longestStreak;
```

```
}
```

```
public static void main(String[] args) {
```

```
    int[] arr = {100, 4, 200, 1, 3, 2};
```

```
    System.out.println("Length of longest subsequence: " + findLongestConseqSubseq(arr,  
arr.length));
```

```
}
```

```
}
```

Output

Length of longest subsequence: 4

=== Code Execution Successful ===

Time Complexity: $O(n)$

Space Complexity: $O(n)$

4. Longest Palindrome in a String

```
class LongestPalindrome {  
    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 = expandFromCenter(s, i, i);  
            int len2 = expandFromCenter(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);
    }

    static int expandFromCenter(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) {
        String s = "babad";
        System.out.println("Longest palindrome: " + longestPalindrome(s));
    }
}

```

Output

Longest palindrome: aba

=== Code Execution Successful ===

Time Complexity: $O(n^2)$

Space Complexity: $O(1)$

5. Rat in a Maze Problem

```
class RatInMaze {
```

```

static boolean isSafe(int[][] maze, int x, int y, int N) {
    return (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1);
}

```

```

static boolean solveMazeUtil(int[][] maze, int x, int y, int[][] sol, int N) {
    if (x == N - 1 && y == N - 1) {
        sol[x][y] = 1;
        return true;
    }

```

```

    if (isSafe(maze, x, y, N)) {
        sol[x][y] = 1;
        if (solveMazeUtil(maze, x + 1, y, sol, N))
            return true;
        if (solveMazeUtil(maze, x, y + 1, sol, N))
            return true;
        sol[x][y] = 0;
        return false;
    }
    return false;
}

```

```

static boolean solveMaze(int[][] maze, int N) {
    int[][] sol = new int[N][N];
    if (!solveMazeUtil(maze, 0, 0, sol, N)) {
        System.out.println("Solution doesn't exist");
        return false;
    }
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            System.out.print(sol[i][j] + " ");

```

```

    }

    System.out.println();
}

return true;
}

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

    int N = 4;

    solveMaze(maze, N);
}
}

```

```

Output
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 1

=== Code Execution Successful ===

```

Time Complexity: $O(2^{(N^2)})$

Space Complexity: $O(N^2)$