1. Anagrams:

Solution:

```java
import java.util.*;

public class Anagram {
    public static boolean isAnagram(String a, String b) {
        if (a.length() != b.length()) return false;
        char[] c1 = a.toCharArray(), c2 = b.toCharArray();
        Arrays.sort(c1);
        Arrays.sort(c2);
        return Arrays.equals(c1, c2);
    }

    public static void main(String[] args) {
        System.out.println(isAnagram("listen", "silent"));
        System.out.println(isAnagram("hello", "world"));
    }
}
```

Output:

```
[Running] cd "/Users/ramyabharathi/Desktop/JAVA/Practiceset3-12_11/" && javac Anagram.java && java Anagram
true
false

[Done] exited with code=0 in 0.311 seconds
```

Time Complexity: O(1)

2. Row with maximum one's

Solution:

```java
import java.util.*;
class MaxOnesRow {
   public static int rowWithMaxOnes(int[][] mat) {
       int r = 0, c = mat[0].length - 1, maxRow = -1;
       while (r < mat.length && c >= 0) {
           if (mat[r][c] == 1) {
               maxRow = r;
               c--;
           }
           else {
               r++;
           }
       }
       return maxRow;
   }
   public static void main(String[] args) {
       int[][] mat = {
           {0, 0, 1, 1},
           {0, 1, 1, 1},
           {0, 0, 0, 1},
           {1, 1, 1, 1}
       };
       System.out.println(rowWithMaxOnes(mat));
   }
}
```

Output:

```
[Running] cd "/Users/ramyabharathi/Desktop/JAVA/Practiceset3-12_11/" && javac MaxOnesRow.java && java MaxOnesRow
3

[Done] exited with code=0 in 0.313 seconds
```

Time Complexity:O(n+m)

3. Longest consequent subsequence:

Solution:

```java
import java.util.*;

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

        int maxLen = 0;

        for (int num : nums) {
            if (!set.contains(num - 1)) {
                int currentNum = num;
                int count = 1;

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

                maxLen = Math.max(maxLen, count);
            }
        }

        return maxLen;
    }

    public static void main(String[] args) {
        int[] nums = {100, 4, 200, 1, 3, 2};
        int[] nums1 ={1, 9, 3, 10, 4, 20, 2};
        System.out.println(longestConsecutive(nums));
        System.out.println(longestConsecutive(nums1));
    }
}
```

Output:

```
[Running] cd "/Users/ramyabharathi/Desktop/JAVA/Practiceset3-12_11/" && javac LongConSub.java && java LongConSub
4
4

[Done] exited with code=0 in 0.332 seconds
```

Time Complexity: O(n)

4.Longest Palindromic Substring

Solution:

```java
import java.util.*;
public class LongPalindrom {
   public static String longestPalindrome(String s) {
       int start = 0, end = 0;
       for (int i = 0; i < s.length(); i++) {
           int len1 = expand(s, i, i);
           int len2 = expand(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 expand(String s, int l, int r) {
       while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
           l--;
           r++;
       }
       return r - l - 1;
   }

   public static void main(String[] args) {
       System.out.println(longestPalindrome("babad"));
       System.out.println(longestPalindrome("abcdedc"));

   }
}
```

Output:

```
[Running] cd "/Users/ramyabharathi/Desktop/JAVA/Practiceset3-12_11/" && javac LongPalindrom.java && java LongPalindrom
aba
cdedc

[Done] exited with code=0 in 0.319 seconds
```

Time Complexity: O(n^2)

---

5. Rat in Maze:

Solution:

```
import java.util.*;
public class Ratinmaze {
   public static ArrayList<String> findPaths(int[][] maze, int n) {
       ArrayList<String> paths = new ArrayList<>();
       if (maze[0][0] == 0 || maze[n - 1][n - 1] == 0) return paths;
       boolean[][] visited = new boolean[n][n];
       solveMaze(maze, n, 0, 0, "", paths, visited);
       Collections.sort(paths);
       return paths;
   }

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

       if (isSafe(maze, n, x, y, visited)) {
           visited[x][y] = true;

           solveMaze(maze, n, x + 1, y, path + 'D', paths, visited);
           solveMaze(maze, n, x - 1, y, path + 'U', paths, visited);
           solveMaze(maze, n, x, y + 1, path + 'R', paths, visited);
           solveMaze(maze, n, x, y - 1, path + 'L', paths, visited);

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

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

   public static void main(String[] args) {
       int[][] maze = {
           {1, 0, 0, 0},
           {1, 1, 0, 1},
           {0, 1, 0, 0},
           {1, 1, 1, 1}
       };
       System.out.println(findPaths(maze, maze.length));
   }
}
```

Output:

```
[Running] cd "/Users/ramyabharathi/Desktop/JAVA/Practiceset3-12_11/" && javac Ratinmaze.java && java Ratinmaze
Maze:
1 0 0 0
1 1 0 1
0 1 0 0
1 1 1 1

[DRDDRR]
```

Time Complexity:O(n)