1.ANAGRAM:

CODE:

```
import java.util.HashMap;
public class Main {
    public static boolean areAnagrams(String s1, String s2) {
        if (s1.length() != s2.length()) {
            return false;
        }
        HashMap<Character, Integer> frequencyMap = new HashMap<>();
        for (int i = 0; i < s1.length(); i++) {
            frequencyMap.put(s1.charAt(i), frequencyMap.getOrDefault(s1.charAt(i), 0) + 1);
            frequencyMap.put(s2.charAt(i), frequencyMap.getOrDefault(s2.charAt(i), 0) - 1);
        }
        for (int value : frequencyMap.values()) {
            if (value != 0) {
                return false;
            }
        }
        return true;
    }
    public static void main(String[] args) {
        String s1 = "geeks";
        String s2 = "kseeg";
        System.out.println(areAnagrams(s1, s2));  // Output: true
    }
}
```

OUTPUT:

```
true

=== Code Execution Successful ===
```

TIME COMPLEXITY: O(n)
SPACE COMPLEXITY: O(1)

2.ROW WITH MAX 1'S:

CODE:

```
public class Main {
   public static int rowWithMax1s(int[][] arr) {
      int n = arr.length;
      int m = arr[0].length;
      int max_row_index = -1;
      int max_ones_count = -1;
      int j = m - 1;
      for (int i = 0; i < n; i++) {
         while (j >= 0 && arr[i][j] == 1) {
            j--;
         }
         int ones_count = m - j - 1;
         if (ones_count > max_ones_count) {
            max_ones_count = ones_count;
            max_row_index = i;
         }
      }
      return max_row_index;
   }
   public static void main(String[] args) {
      int[][] arr = {
         {0, 1, 1, 1},
         {0, 0, 1, 1},
         {1, 1, 1, 1},
         {0, 0, 0, 0}
      };
      System.out.println(rowWithMax1s(arr));
   }
}
```

OUTPUT:

```
2

=== Code Execution Successful ===
```

TIME COMPLEXITY: O(n + m)

SPACE COMPLEXITY: O(1)

3. Longest Consecutive Subsequence

CODE:

```
import java.util.HashSet;
public class Main {
    public static int longestConsecutiveSubsequence(int[] arr) {
        if (arr.length == 0) return 0;
        HashSet<Integer> set = new HashSet<>();
        for (int num : arr) {
            set.add(num);
        }
        int maxLength = 0;
        for (int num : arr) {
            if (!set.contains(num - 1)) {
                int currentNum = num;
                int currentLength = 1;
                while (set.contains(currentNum + 1)) {
                    currentNum++;
                    currentLength++;
                }
                maxLength = Math.max(maxLength, currentLength);
            }
        }
        return maxLength;
    }
    public static void main(String[] args) {
        int[] arr1 = {2, 6, 1, 9, 4, 5, 3};
        int[] arr2 = {1, 9, 3, 10, 4, 20, 2};
        int[] arr3 = {15, 13, 12, 14, 11, 10, 9};
        System.out.println(longestConsecutiveSubsequence(arr1));
        System.out.println(longestConsecutiveSubsequence(arr2));
        System.out.println(longestConsecutiveSubsequence(arr3));
    }
}
```

OUTPUT:

```
6
4
7

=== Code Execution Successful ===
```

TIME COMPLEXITY: O(n)
SPACE COMPLEXITY: O(n)

4. Longest palindrome in a string

CODE:

```
public class Main {
    public static String longestPalindrome(String s) {
        if (s == null || s.length() == 0) {
            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;
    }
    public static void main(String[] args) {
        System.out.println(longestPalindrome("aaaabbaa"));
        System.out.println(longestPalindrome("abc"));
        System.out.println(longestPalindrome("abacdfgdcaba"));
    }
}
```

OUTPUT:

```
aabbaa
c
aba

=== Code Execution Successful ===
```

TIME COMPLEXITY: O(n^2)

SPACE COMPLEXITY: O(1)

5. Rat in a maze problem

CODE:

```
import java.util.ArrayList;
import java.util.Collections;
public class Main {
   static boolean isSafe(int[][] mat, int x, int y, int n, boolean[][] visited) {
      return (x >= 0 && x < n) && (y >= 0 && y < n) && mat[x][y] == 1 && !visited[x][y];
   }
   static void findPathsUtil(int[][] mat, int x, int y, int n, boolean[][] visited, String path,
ArrayList<String> paths) {
      if (x == n - 1 && y == n - 1) {
         paths.add(path);
         return;
      }
      visited[x][y] = true;
      if (isSafe(mat, x + 1, y, n, visited)) {
         findPathsUtil(mat, x + 1, y, n, visited, path + 'D', paths);
      }
      if (isSafe(mat, x, y - 1, n, visited)) {
         findPathsUtil(mat, x, y - 1, n, visited, path + 'L', paths);
      }
      if (isSafe(mat, x, y + 1, n, visited)) {
         findPathsUtil(mat, x, y + 1, n, visited, path + 'R', paths);
      }
      if (isSafe(mat, x - 1, y, n, visited)) {
         findPathsUtil(mat, x - 1, y, n, visited, path + 'U', paths);
      }
      visited[x][y] = false;
   }
   static ArrayList<String> findPaths(int[][] mat, int n) {
      boolean[][] visited = new boolean[n][n];
      if (mat[0][0] == 0) {
         return new ArrayList<>();
      }
      ArrayList<String> paths = new ArrayList<>();
      findPathsUtil(mat, 0, 0, n, visited, "", paths);
      Collections.sort(paths);
      return paths;
   }
   public static void main(String[] args) {
      int[][] mat1 = {
         {1, 0, 0, 0},
         {1, 1, 0, 1},
         {1, 1, 0, 0},
```

```
        {0, 1, 1, 1}
    };
    System.out.println(findPaths(mat1, 4));
    int[][] mat2 = {
        {1, 0},
        {1, 0}
    };
    System.out.println(findPaths(mat2, 2));
    int[][] mat3 = {
        {1, 1, 1},
        {1, 0, 1},
        {1, 1, 1}
    };
    System.out.println(findPaths(mat3, 3));
  }
}
```

OUTPUT:

```
[DDRDRR, DRDDRR]
[]
[DDRR, RRDD]

=== Code Execution Successful ===
```