CATHERINE JESSIE J
22CS035 CSE

1) Anagram Program

```
import java.util.HashMap;

class Anagram {
    public static void main(String[] args) {
        ans("geeks", "kseeg");
        ans("allergy", "allergic");
        ans("g", "g");
    }

    public static void ans(String s1, String s2) {
        if (s1.length() != s2.length()) {
            System.out.println("false");
            return;
        }

        HashMap<Character, Integer> h1 = new HashMap<>();
        for (char c : s1.toCharArray()) {
            h1.put(c, h1.getOrDefault(c, 0) + 1);
        }

        HashMap<Character, Integer> h2 = new HashMap<>();
        for (char c : s2.toCharArray()) {
            h2.put(c, h2.getOrDefault(c, 0) + 1);
        }

        if (h1.equals(h2)) {
            System.out.println("true");
        } else {
            System.out.println("false");
        }
    }
}
```

Time Complexity: O(n)
Space Complexity: O(n)

2) Row with Maximum 1s

```
class Solution {
    public int rowWithMax1s(int arr[][]) {
        int r = arr.length, c = arr[0].length, maxRow = -1, j = c - 1;
        for (int i = 0; i < r; i++) {
            while (j >= 0 && arr[i][j] == 1) {
                j--;
                maxRow = i;
```

```
            }
        }
        return maxRow;
    }
}
```

Time Complexity: O(r+c)
Space Complexity: O(1)

3) Longest Consecutive Subsequence

```
class Solution {

    public int findLongestConseqSubseq(int[] arr) {
        Arrays.sort(arr);
        int max = 0, count = 1;
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] == arr[i - 1]) {
                continue;
            }
            if (arr[i] == arr[i - 1] + 1) {
                count++;
            } else {
                max = Math.max(max, count);
                count = 1;
            }
        }
        return Math.max(max, count);
    }
}
```

Time Complexity: O(n log n)
Space Complexity: O(n)

4) Longest Palindrome in a String

```
class Solution {
    static String longestPalindrome(String s) {
        int l = 0, r = 0;
        for (int i = 0; i < s.length(); i++) {
            int l1 = i, r1 = i;
            while (l1 >= 0 && r1 < s.length() && s.charAt(l1) == s.charAt(r1)) {
                l1--;
                r1++;
            }
            int l2 = i, r2 = i + 1;
            while (l2 >= 0 && r2 < s.length() && s.charAt(l2) == s.charAt(r2)) {
                l2--;
```

```
            r2++;
        }
        if (r1 - l1 - 1 > r - l) {
            l = l1 + 1;
            r = r1;
        }
        if (r2 - l2 - 1 > r - l) {
            l = l2 + 1;
            r = r2;
        }
    }
    return s.substring(l, r);
  }
}
```

Time Complexity: O(n^2)
Space Complexity: O(1)

5) Rat in a Maze Problem

```
class Solution {
  public ArrayList<String> findPath(int[][] mat) {
    ArrayList<String> res = new ArrayList<>();
    int n = mat.length;
    int m = mat[0].length;
    if (mat[0][0] == 0) return res;
    dfs(mat, 0, 0, n, m, "", res);
    return res;
  }

  public void dfs(int[][] mat, int i, int j, int n, int m, String path, ArrayList<String> res) {
    if (i == n - 1 && j == m - 1) {
      res.add(path);
      return;
    }
    mat[i][j] = 0;
    if (i + 1 < n && mat[i + 1][j] == 1) dfs(mat, i + 1, j, n, m, path + "D", res);
    if (j - 1 >= 0 && mat[i][j - 1] == 1) dfs(mat, i, j - 1, n, m, path + "L", res);
    if (j + 1 < m && mat[i][j + 1] == 1) dfs(mat, i, j + 1, n, m, path + "R", res);
    if (i - 1 >= 0 && mat[i - 1][j] == 1) dfs(mat, i - 1, j, n, m, path + "U", res);
    mat[i][j] = 1;
  }
}
```

Time Complexity: O(2^(N^2))
Space Complexity: O(N^2)

CATHERINE JESSIE J
22CS035 CSE