Day - 2 : 12/11/2024

1. **Anagram:**

```
class Solution {
public:
    bool isAnagram(string str1, string str2) {
        if (str1.length() != str2.length()) return false;
        sort(str1.begin(), str1.end());
        sort(str2.begin(), str2.end());

        for (int i = 0; i < str1.length(); i++) {
            if (str1[i] != str2[i]) return false;
        }

        return true;
    }
};
```

Time Complexity: O(n log n)

2. **Row with Max 1's:**

```
class Solution {
public:
    int rowWithMax1s(vector<vector<int>>& matrix) {
        int resultRow = -1, maxOnes = 0;
        int rows = matrix.size(), cols = matrix[0].size();
        int colIndex = cols - 1;
        for (int i = 0; i < rows; i++) {
            while (colIndex >= 0 && matrix[i][colIndex] == 1) {
                resultRow = i;
                colIndex--;
            }
        }
        return resultRow;
    }
};
```

Time Complexity: O(m *n)

3. **Longest Consecutive Subsequence:**

```
class Solution {
public:
    int findLongestConseqSubseq(vector<int>& nums) {
        if (nums.empty()) return 0;
```

```
        sort(nums.begin(), nums.end());
        int longestSeq = 1, currentSeq = 1;
        for (int i = 1; i < nums.size(); i++) {
            if (nums[i] == nums[i - 1]) continue;
            if (nums[i] == nums[i - 1] + 1) {
                currentSeq++;
            } else {
                longestSeq = max(longestSeq, currentSeq);
                currentSeq = 1;
            }
        }

        return max(longestSeq, currentSeq);
    }
};
```

Time Complexity: O(n log n)

4. **Longest Palindromic Substring:**

```
class Solution {
public:
    string longestPalindrome(string text) {
        int startPos = 0, longestLength = 1;
        for (int i = 0; i < text.length(); i++) {
            expandAroundCenter(text, i, i, startPos, longestLength);
            expandAroundCenter(text, i, i + 1, startPos, longestLength);
        }
        return text.substr(startPos, longestLength);
    }
private:
    void expandAroundCenter(const string& text, int left, int right, int& startPos, int& longestLength)
{
        while (left >= 0 && right < text.length() && text[left] == text[right]) {
            if (right - left + 1 > longestLength) {
                startPos = left;
                longestLength = right - left + 1;
            }
            left--;
            right++;
        }
    }
};
```

Time Complexity: O(N^2)

5. **Rat in a Maze:**

```
class Solution {
public:
    bool solveMaze(vector<vector<int>>& maze) {
        vector<vector<int>> solutionPath(maze.size(), vector<int>(maze[0].size(), 0));
        return findPath(maze, 0, 0, solutionPath);
    }
private:
    bool findPath(vector<vector<int>>& maze, int row, int col, vector<vector<int>>& solutionPath) {
        int rows = maze.size();
        int cols = maze[0].size();
        if (row == rows - 1 && col == cols - 1 && maze[row][col] == 1) {
            solutionPath[row][col] = 1;
            displaySolution(solutionPath);
            return true;
        }
        if (canMove(maze, row, col)) {
            solutionPath[row][col] = 1;

            if (findPath(maze, row + 1, col, solutionPath)) return true;
            if (findPath(maze, row, col + 1, solutionPath)) return true;

            solutionPath[row][col] = 0;
        }
        return false;
    }
    bool canMove(vector<vector<int>>& maze, int row, int col) {
        return (row >= 0 && row < maze.size() && col >= 0 && col < maze[0].size() &&
maze[row][col] == 1);
    }
    void displaySolution(vector<vector<int>>& solutionPath) {
        for (const auto& row : solutionPath) {
            for (int cell : row) {
                cout << cell << " ";
            }
            cout << endl;
        }
    }
};
```

Time Complexity: O(2^n*m)