Longest Nice Substring

```
class Solution {
public:
    string longestNiceSubstring(string s) {

        if (s.length() < 2) return ""; // A single character cannot be nice

        unordered_set<char> charSet(s.begin(), s.end());

        // Find the first character that doesn't satisfy the "nice" condition
        for (int i = 0; i < s.size(); i++) {
            char ch = s[i];
            if (charSet.count(tolower(ch)) && charSet.count(toupper(ch))) {
                continue; // This character is fine
            }

            // Recursively solve left and right substrings
            string left = longestNiceSubstring(s.substr(0, i));
            string right = longestNiceSubstring(s.substr(i + 1));

            // Return the longer one
            return left.size() >= right.size() ? left : right;
        }

        return s; // If entire string is nice, return it

    }
};
```

Maximum Subarray

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
         int maxSum = INT_MIN, currentSum = 0;

        for (int num : nums) {
            currentSum = max(num, currentSum + num); // Choose the best option
            maxSum = max(maxSum, currentSum);       // Update max sum
        }

        return maxSum;
    }

};
```

```cpp
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {

        int m = matrix.size(), n = matrix[0]×size();
        int row = 0, col = n - 1; // Start at top-right corner

        while (row < m && col >= 0) {
            if (matrix[row][col] == target) {
                return true; // Found the target
            } else if (matrix[row][col] > target) {
                col--; // Move left
            } else {
                row++; // Move down
            }
        }

        return false; // Not found
    }
};
```