Name: Ishan Sharma

UID: 22BCS11144

Sec/Group = KPIT 901-B

Problem 1:

```cpp
#include <iostream>
using namespace std;

// Function to calculate Fibonacci number using recursion
int fibonacci(int n) {
    if (n == 0) {
        return 0; // Base case: F(0) = 0
    } else if (n == 1) {
        return 1; // Base case: F(1) = 1
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2); // Recursive case
    }
}

int main() {
    int n;
    cout << "Enter the value of n: ";
    cin >> n;

    // Validate input
    if (n < 0 || n > 30) {
        cout << "Invalid input. Please enter a value between 0 and 30.\n";
```

input

```
Enter the value of n: 12
F(12) = 144


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 2:

```cpp
#include <iostream>
using namespace std;

// Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode* next;

    ListNode(int x) : val(x), next(nullptr) {}
};

// Function to reverse the linked list iteratively
ListNode* reverseListIterative(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* curr = head;
    while (curr != nullptr) {
        ListNode* nextTemp = curr->next; // Save the next node
        curr->next = prev;               // Reverse the link
        prev = curr;                     // Move prev forward
        curr = nextTemp;                 // Move curr forward
    }
    return prev;
}

// Function to reverse the linked list recursively
ListNode* reverseListRecursive(ListNode* head) {
    if (head == nullptr || head->next == nullptr) {
        return head; // Base case: empty list or single node
```

input

```
Original List: 1 2 3 4 5
Reversed List (Iterative): 5 4 3 2 1
Original List: 1 2
Reversed List (Recursive): 2 1
```

Problem 3:

```cpp
#include <iostream>
using namespace std;

// Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

// Function to add two numbers represented by linked lists
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    ListNode* dummyHead = new ListNode(0); // Dummy node to start the result list
    ListNode* current = dummyHead;         // Pointer to traverse the result list
    int carry = 0;                         // Initialize carry to 0

    // Traverse both lists until both are null
    while (l1 != nullptr || l2 != nullptr || carry != 0) {
        int sum = carry; // Start with the carry
        if (l1 != nullptr) {
            sum += l1->val;
            l1 = l1->next;
        }
        if (l2 != nullptr) {
            sum += l2->val;
            l2 = l2->next;
        }
        carry = sum / 10;            // Calculate carry
```

input

```
Input l2: 0
Output: 0
Input l1: 9 -> 9 -> 9 -> 9 -> 9 -> 9 -> 9
Input l2: 9 -> 9 -> 9 -> 9
Output: 8 -> 9 -> 9 -> 9 -> 0 -> 0 -> 0 -> 1
```

Problem 4:

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

// Function to implement wildcard matching
bool isMatch(string s, string p) {
    int m = s.size(), n = p.size();
    vector<vector<bool>> dp(m + 1, vector<bool>(n + 1, false));

    // Base case: Empty string matches with empty pattern
    dp[0][0] = true;

    // Handle patterns with '*'
    for (int j = 1; j <= n; ++j) {
        if (p[j - 1] == '*') {
            dp[0][j] = dp[0][j - 1];
        }
    }

    // Fill the DP table
    for (int i = 1; i <= m; ++i) {
        for (int j = 1; j <= n; ++j) {
            if (p[j - 1] == s[i - 1] || p[j - 1] == '?') {
                dp[i][j] = dp[i - 1][j - 1]; // Characters match or '?' matches any sir
            } else if (p[j - 1] == '*') {
                dp[i][j] = dp[i - 1][j] || dp[i][j - 1]; // '*' matches zero or more ch
            }
```

input

```
Input: s = "aa", p = "a"
Output: false
Input: s = "aa", p = "*"
Output: true
Input: s = "cb", p = "?a"
Output: false
```

Problem 5:

```cpp
#include <algorithm>
using namespace std;

// Function to calculate GCD using the Euclidean algorithm
int findGCD(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

// Main function to find the GCD of the smallest and largest numbers in the array
int gcdOfArray(const vector<int>& nums) {
    int smallest = *min_element(nums.begin(), nums.end());
    int largest = *max_element(nums.begin(), nums.end());
    return findGCD(smallest, largest);
}

int main() {
    // Example 1
    vector<int> nums1 = {2, 5, 6, 9, 10};
    cout << "Input: nums = [2, 5, 6, 9, 10]" << endl;
    cout << "Output: " << gcdOfArray(nums1) << endl;

    // Example 2
    vector<int> nums2 = {7, 5, 6, 8, 3};
```

input

```
Input: nums = [2, 5, 6, 9, 10]
Output: 2
Input: nums = [7, 5, 6, 8, 3]
Output: 1
Input: nums = [3, 3]
Output: 3
```