Department of Computer Science and Engineering

Name:- Vikash Ranjan Kumar UID:- 22BCS11322 Section:- 22KPIT-901-B

DAY-3

Q.1 .**Fibonacci Series Using Recursion**

The Fibonacci numbers, commonly denoted F(n) form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

F(0) = 0, F(1) = 1

F(n) = F(n - 1) + F(n - 2), for n > 1.

Given n, calculate F(n).

Example 1:

Input: n = 2

Output: 1

Explanation: F(2) = F(1) + F(0) = 1 + 0 = 1

Example 2:

Input: n = 3

Output: 2

Explanation: F(3) = F(2) + F(1) = 1 + 1 = 2.

Program Code:-

```
#include <iostream>
using namespace std;
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}
int main() {
    int n;
    cout << "Enter the value of n: ";
    cin >> n;
    cout << "Fibonacci number F(" << n << ") = " << fibonacci(n) << endl;
    return 0;
}
```

Output:-

```
Enter the value of n: 6
Fibonacci number F(6) = 8
```

Q.2 . **Reverse Linked List**

Given the head of a singly linked list, reverse the list, and return the reversed list.

Example 1:

Input: head = [1,2,3,4,5]

Output: [5,4,3,2,1]

Example 2:

Input: head = [1,2]

Output: [2,1]

Program Code:-

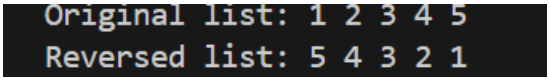
```
#include <iostream>
using namespace std;

struct ListNode {
    int val;
```

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

ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* current = head;

    while (current != nullptr) {
        ListNode* nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    }
    return prev;
}

void printList(ListNode* head) {
    while (head != nullptr) {
        cout << head->val << " ";
```

```
        head = head->next;
    }
    cout << endl;
}

ListNode* createList(const int arr[], int size) {
    if (size == 0) return nullptr;
    ListNode* head = new ListNode(arr[0]);
    ListNode* current = head;
    for (int i = 1; i < size; ++i) {
        current->next = new ListNode(arr[i]);
        current = current->next;
    }
    return head;
}

int main() {
    int arr1[] = {1, 2, 3, 4, 5};
    int size1 = sizeof(arr1) / sizeof(arr1[0]);
```

```
    ListNode* head1 = createList(arr1, size1);
    cout << "Original list: ";
    printList(head1);

    ListNode* reversedHead1 = reverseList(head1);
    cout << "Reversed list: ";
    printList(reversedHead1);

    return 0;
}
```

Output:-



Q.3. **.: Longest Substring Without Repeating Characters**

You are working on building a text editor application. One of the features you're focusing on is a "word suggestion" system. The editor needs to identify the longest sequence of characters typed without repeating any letters to suggest potential words or phrases. To accomplish this, you must efficiently find the length of the longest substring of unique characters as the user types.

Description:
Write a function that takes a string as input and returns the length of the longest

substring without repeating characters. A substring is a contiguous sequence of characters within the string.

Example 1:

Input: "abcabcbb"

Output: 3

Explanation: The longest substring without repeating characters is "abc", which has length 3.

Constraints:

- The input string will have a length between 111 and 10410^4104.
- The characters in the string are printable ASCII characters.

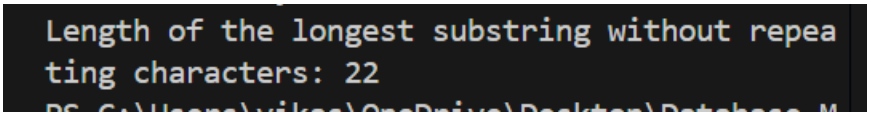
This problem is a classic example of the **sliding window** technique,

Program Code:-

```
#include <iostream>

#include <string>

#include <unordered_set>

using namespace std;

int findLongestSubstring(const string& s, int start, int end, unordered_set<char>& seen) {

    if (end == s.length()) {

        return 0;
```

```
    }

    if (seen.find(s[end]) == seen.end()) {
        seen.insert(s[end]);
        return 1 + findLongestSubstring(s, start, end + 1, seen);
    } else {
        seen.clear();
        return max(findLongestSubstring(s, start + 1, start + 1, seen),
                   findLongestSubstring(s, start, end + 1, seen));
    }
}

int lengthOfLongestSubstring(string s) {
    unordered_set<char> seen;
    return findLongestSubstring(s, 0, 0, seen);
}

int main() {
    string input = "abcabcbb";
    cout << "Length of the longest substring without repeating characters: "
         << lengthOfLongestSubstring(input) << endl;
    return 0;
}
```

Output:-



Q.4

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1:

Input: l1 = [2,4,3], l2 = [5,6,4]

Output: [7,0,8]

Explanation: 342 + 465 = 807.

Example 2:

Input: l1 = [0], l2 = [0]

Output: [0]

Example 3:

Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

Output: [8,9,9,9,0,0,0,1]

Constraints:

The number of nodes in each linked list is in the range [1, 100].

0 <= Node.val <= 9

It is guaranteed that the list represents a number that does not have leading zeros.

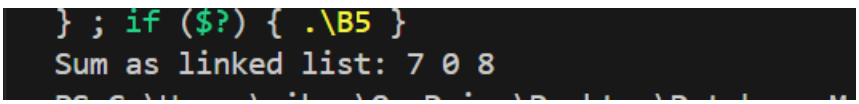
Program Code:-

```
#include <iostream>
using namespace std;
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    ListNode* dummyHead = new ListNode(0);
    ListNode* current = dummyHead;
    int carry = 0;
    while (l1 != nullptr || l2 != nullptr || carry != 0) {
        int sum = carry;
        if (l1 != nullptr) {
            sum += l1->val;
```

```
            l1 = l1->next;
        }
        if (l2 != nullptr) {
            sum += l2->val;
            l2 = l2->next;
        }
        carry = sum / 10;
        current->next = new ListNode(sum % 10);
        current = current->next;
    }
    return dummyHead->next;
}
void printList(ListNode* head) {
    while (head != nullptr) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}

ListNode* createList(const int arr[], int size) {
    if (size == 0) return nullptr;
```

```
    ListNode* head = new ListNode(arr[0]);
    ListNode* current = head;
    for (int i = 1; i < size; ++i) {
        current->next = new ListNode(arr[i]);
        current = current->next;
    }
    return head;
}
int main() {
    int arr1[] = {2, 4, 3};
    int arr2[] = {5, 6, 4};
    int size1 = sizeof(arr1) / sizeof(arr1[0]);
    int size2 = sizeof(arr2) / sizeof(arr2[0]);
    ListNode* l1 = createList(arr1, size1);
    ListNode* l2 = createList(arr2, size2);
    ListNode* result = addTwoNumbers(l1, l2);
    cout << "Sum as linked list: ";
    printList(result);
    return 0;
}
```

Output:-



Q.5. You are developing a financial analysis application where users input daily stock price changes in an array. Your task is to determine the maximum profit or loss achievable over a contiguous period, considering that the profit/loss for a period is calculated by multiplying the daily percentage changes. This problem is critical for finding optimal periods to make decisions for investments or short-term trading.

Description:

Write a function that takes an integer array as input and returns the maximum product of a contiguous subarray. The array can contain both positive and negative integers, and your function must account for these to find the optimal subarray.

Input:

- An array of integers nums[]nums[]nums[] where 1≤nums.length≤1041 \leq nums.length \leq 10^41≤nums.length≤104 and −10≤nums[i]≤10-10 \leq nums[i] \leq 10−10≤nums[i]≤10.

Output:

- An integer representing the maximum product of any contiguous subarray.

Example 1:

Input: nums = [2, 3, -2, 4]
Output: 6
Explanation: The subarray [2, 3] has the largest product = 6.

Constraints:

1. The input array may contain both positive and negative numbers, including zero.
2. You must solve the problem with a time complexity of O(n)O(n)O(n).

Program code –

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;



int maxProduct(vector<int>& nums) {
```

```
    if (nums.empty()) return 0;

    int maxProd = nums[0];
    int minProd = nums[0];
    int result = nums[0];

    for (int i = 1; i < nums.size(); ++i) {
        if (nums[i] < 0) {
            swap(maxProd, minProd);
        }

        maxProd = max(nums[i], maxProd * nums[i]);
        minProd = min(nums[i], minProd * nums[i]);

        result = max(result, maxProd);
    }

    return result;
}

int main() {
    vector<int> nums = {2, 3, -2, 4};
```

```
    cout << "Maximum product of a contiguous subarray: " << maxProduct(nums)
<< endl;

    return 0;

}
```

Output:-

```
Maximum product of a contiguous subarray: 6
```