DAY 2

Student Name: Shivam   **UID: 22BCS15301**
Branch: BE-CSE   **Section/Group: 620 - A**
Date of Performance:20/12/24

Problem 1

1. **Aim: Majority Elements**
2. **Problem Statement:** Given an array nums of size n, return the majority element.

   The majority element is the element that appears more than ⌊ n / 2⌋ times. You may assume that the majority element always exists in the array.

3. **Code:**

```cpp
#include<iostream>
using namespace std;
int main()
{
    int arr[5] = {1, 2, 3, 3, 5};
    int ans[5] = {0};
    int count = 1, value, max;

    for (int i = 0; i < 5; i++)
    {
        ans[arr[i]-1]++;
    }
    for (int i = 0; i < 5; i++)
    {
        if (ans[i]>ans[i+1])
        {
            max = ans[i];
```

```
        }
      }
      for (int i = 0; i < 5; i++)
      {
          cout<<i+1<<" : "<<ans[i]<<endl;
      }
      return 0;
  }
```

4. Output:

```
1 : 1
2 : 1
3 : 2
4 : 0
5 : 1
```

Problem 2

1. Aim: Pascal's Triangle

2. Problem Statement: Given an integer numRows, return the first numRows of Pascal's triangle.
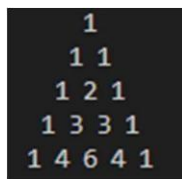In Pascal's triangle, each number is the sum of the two numbers directly above it as shown.

3. Code:

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

int main()
{
   int n = 5;
    for (int i = 0; i < n; i++) {
       int value = 1;
```

```cpp
        for (int j = 0; j < n - i - 1; j++) {
            cout << " ";
        }
        for (int j = 0; j <= i; j++) {
            cout << value << " ";
            value = value * (i - j) / (j + 1);
        }
        cout << endl;
    }
    return 0;
}
```

4. Output:


```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Problem 3

1. Aim: Single Number

2. Problem Statement: Given a non-empty array of integers nums, every element appears twice except for one. Find that single one. You must implement a solution with a linear runtime complexity and use only constant extra space.

3. Code:

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

int singleNumber(const vector<int>& nums) {
    int result = 0;
```

```cpp
    for (int num : nums) {
        result ^= num;
    }
    return result;
}
int main() {
    std::vector<int> nums = {2, 2, 1};
    std::cout << "The single number is: " << singleNumber(nums) << std::endl;
    return 0;
}
```

4. **Output:**

```
The single number is: 1
```

Problem 4

1. **Aim: Merge Two Sorted Lists**

2. **Problem Statement:** You are given the heads of two sorted linked lists list1 and list2.
   Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.
   Return the head of the merged linked list

3. **Code:**

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

struct ListNode {
```

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

ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    ListNode dummy(0);
    ListNode* tail = &dummy;
    while (list1 != nullptr && list2 != nullptr) {
        if (list1->val < list2->val) {
            tail->next = list1;
            list1 = list1->next;
        } else {
            tail->next = list2;
            list2 = list2->next;
        }
        tail = tail->next;
    }
    if (list1 != nullptr) {
        tail->next = list1;
    } else {
        tail->next = list2;
    }
    return dummy.next;
}
void printList(ListNode* head) {
    while (head != nullptr) {
        std::cout << head->val << " ";
        head = head->next;
    }
    std::cout << std::endl;
}
int main() {
    ListNode* list1 = new ListNode(1);
    list1->next = new ListNode(2);
```

```
        list1->next->next = new ListNode(4);
        ListNode* list2 = new ListNode(1);
        list2->next = new ListNode(3);
        list2->next->next = new ListNode(4);
        ListNode* mergedList = mergeTwoLists(list1, list2);
        printList(mergedList);
        return 0;
    }
```

4. Output:

```
1 1 2 3 4 4
```

Problem 5

1. Aim: Remove Linked List Elements

2. Problem Statement: Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return *the new head*.

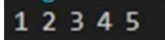## 3. Code:

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

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

```cpp
class Solution {
public:
    ListNode* removeElements(ListNode* head, int val) {
        ListNode dummy(0);
        dummy.next = head;
        ListNode* current = &dummy;
        while (current->next != nullptr) {
            if (current->next->val == val) {
                current->next = current->next->next;
            } else {
                current = current->next;
            }
        }
        return dummy.next;
    }
};
ListNode* createLinkedList(const std::vector<int>& values) {
    ListNode* head = nullptr;
    ListNode* tail = nullptr;
    for (int value : values) {
        ListNode* newNode = new ListNode(value);
        if (!head) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}
void printLinkedList(ListNode* head) {
    ListNode* current = head;
    while (current) {
        std::cout << current->val << " ";
```

```cpp
        current = current->next;
    }
    std::cout << std::endl;
}
int main() {
    Solution solution;
    std::vector<int> values = {1, 2, 6, 3, 4, 5, 6};
    int val = 6;
    ListNode* head = createLinkedList(values);
    ListNode* newHead = solution.removeElements(head, val);
    printLinkedList(newHead);
    return 0;
}
```

4. **Output:**

```
1 2 3 4 5
```

Problem 6

1. **Aim: Reverse Linked List**
2. **Problem Statement:** Given the head of a singly linked list, reverse the list, and return *the reversed list*.
3. **Code:**

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

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

class Solution {
```

```cpp
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* current = head;
        ListNode* next = nullptr;
        while (current != nullptr) {
            next = current->next;
            current->next = prev;
            prev = current;
            current = next;
        }
        return prev;
    }
};
void printList(ListNode* head) {
    ListNode* current = head;
    while (current != nullptr) {
        std::cout << current->val << " ";
        current = current->next;
    }
    std::cout << std::endl;
ListNode* createList(const std::vector<int>& values) {
    if (values.empty()) return nullptr;
    ListNode* head = new ListNode(values[0]);
    ListNode* current = head;
    for (size_t i = 1; i < values.size(); ++i) {
        current->next = new ListNode(values[i]);
        current = current->next;
    }
    return head;
}
int main() {
    Solution solution;
    ListNode* head = createList({1, 2, 3, 4, 5});
    std::cout << "Original list: ";
```

```
    printList(head);
    ListNode* reversedHead = solution.reverseList(head);
    std::cout << "Reversed list: ";
    printList(reversedHead);
    return 0;
}
```

4. Output:

```
INPUT NO. : 5
9
```

Problem 7

1. **Aim: Reverse a Number**
2. **Problem Statement:** Reverse the digits of a given number n. For example, if the input number is 12345, the output should be 54321. The task involves using loops and modulus operators to extract the digits and construct the reversed number
3. **Task:** Given an integer n, print the number with its digits in reverse order.
4. **Code:**

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

int main() {
    int num, rev_num = 0;
    cout << "Enter a number: ";
    cin >> num;
    while (num != 0) {
        rev_num = rev_num * 10 + num % 10;
        num /= 10;
```

```
    }
    cout << "Reversed Number: " << rev_num << endl;
    return 0;
}
```

5. **Output:**

```
Original list: 1 2 3 4 5
Reversed list: 5 4 3 2 1
```

Problem 8

1. **Aim: Container With Most Water**
2. **Problem Statement:** You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).
   Find two lines that together with the x-axis form a container, such that the container contains the most water.
   Return the maximum amount of water a container can store
3. **Code:**

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int maxArea(vector<int>& height) {
    int left = 0;
    int right = height.size() - 1;
    int max_area = 0;
    while (left < right) {
        int width = right - left;
        int h = min(height[left], height[right]);
        int area = width * h;
```

```
        max_area = max(max_area, area);
        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }

    return max_area;
}
int main() {
    vector<int> height = {1, 8, 6, 2, 5, 4, 8, 3, 7};
    cout << "Maximum amount of water a container can store: " <<
maxArea(height) << endl;
    return 0;
}
```

4. **Output:**

```
Maximum amount of water a container can store: 49
```

Problem 9

1. **Aim: Jump Game II.**
2. **Problem Statement:** You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0].

   Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

   $0 <= j <= nums[i]$ ani $+ j < n$

Return the minimum number of jumps to reach nums[n - 1]. The test cases are generated such that you can reach nums[n - 1.

3. Code:

```cpp
#include <vector>
#include <iostream>
#include <limits.h>
using namespace std;
int jump(vector<int>& nums) {
    int n = nums.size();
    if (n <= 1) return 0;

    int jumps = 0;
    int currentEnd = 0;
    int farthest = 0;
    for (int i = 0; i < n - 1; i++) {
        farthest = max(farthest, i + nums[i]);
        if (i == currentEnd) {
            jumps++;
            currentEnd = farthest;
            if (currentEnd >= n - 1) break;
        }
    }
    return jumps;
}
int main() {
    vector<int> nums = {2, 3, 1, 1, 4};
    cout << "Minimum number of jumps: " << jump(nums) << endl;
    return 0;
}
```

4. Output:

```
Minimum number of jumps: 2
```

Problem 10

1. **Aim: Linked List Cycle.**
2. **Problem Statement:** Given head, the head of a linked list, determine if the linked list has a cycle in it.

   There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.

   Return true if there is a cycle in the linked list. Otherwise, return false.
3. **Code:**

```cpp
#include <iostream>
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};
bool hasCycle(ListNode *head) {
    if (head == nullptr) return false;
    ListNode *slow = head;
    ListNode *fast = head;
    while (fast != nullptr && fast->next != nullptr) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) {
            return true;
        }
    }
```

```cpp
        return false;
    }
    int main() {
        ListNode *head = new ListNode(3);
        ListNode *second = new ListNode(2);
        ListNode *third = new ListNode(0);
        ListNode *fourth = new ListNode(-4);
        head->next = second;
        second->next = third;
        third->next = fourth;
        fourth->next = second;
        if (hasCycle(head)) {
            std::cout << "The linked list has a cycle." << std::endl;
        } else {
            std::cout << "The linked list does not have a cycle." << std::endl;
        }
        return 0;
    }
```

4. Output:

```
The linked list has a cycle.
```