C++ Week 12

Khaula Molapo

1. Implementing a Doubly Linked List

A Doubly Linked List is a data structure where each node contains data, a next pointer, and a previous (prev) pointer. This allows movement in both forward and backward directions.

Example C++ implementation:

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

struct Node {
   int data;
   Node* next;
   Node* prev;
   Node(int val) : data(val), next(nullptr), prev(nullptr) {}
};

class DoublyLinkedList {
   Node* head;
   Node* tail;
public:
   DoublyLinkedList() : head(nullptr), tail(nullptr) {}

   void append(int val) {
      Node* newNode = new Node(val);
      if (!head) head = tail = newNode;
      else {
         tail->next = newNode;
         newNode->prev = tail;
         tail = newNode;
      }
   }

   void prepend(int val) {
      Node* newNode = new Node(val);
      if (!head) head = tail = newNode;
      else {
         newNode->next = head;
         head->prev = newNode;
         head = newNode;
```

```
    }
  }

  void displayForward() {
    Node* temp = head;
    while (temp) {
      cout << temp->data << " ";
      temp = temp->next;
    }
    cout << endl;
  }

  void displayBackward() {
    Node* temp = tail;
    while (temp) {
      cout << temp->data << " ";
      temp = temp->prev;
    }
    cout << endl;
  }
};
```

Advantages:
- Easy reverse traversal.
- Deleting from both ends is simple.

Disadvantages:
- Uses more memory because of the extra prev pointer.
- More complex than a singly linked list.

2. Performance Analysis

Insertion at the head of a linked list is O(1) because only pointers change. Insertion at the tail is also O(1) when using a tail pointer, but becomes O(n) when the list must be traversed first.

Searching in a linked list is O(n) because each node must be visited. In a dynamic array (std::vector), direct access using an index is O(1), but inserting or deleting elements in the middle can be O(n) since elements may need to shift.

A linked list is better when the number of elements changes a lot and when many insertions or deletions happen at the beginning or end. It is not ideal when fast random access is needed.