Board Infinity Minor Project

Merge Two Sorted Linked Lists

Submitted by:

Tuba Mirza

12319516

Lovely Professional University

Table of Contents

1. Introduction

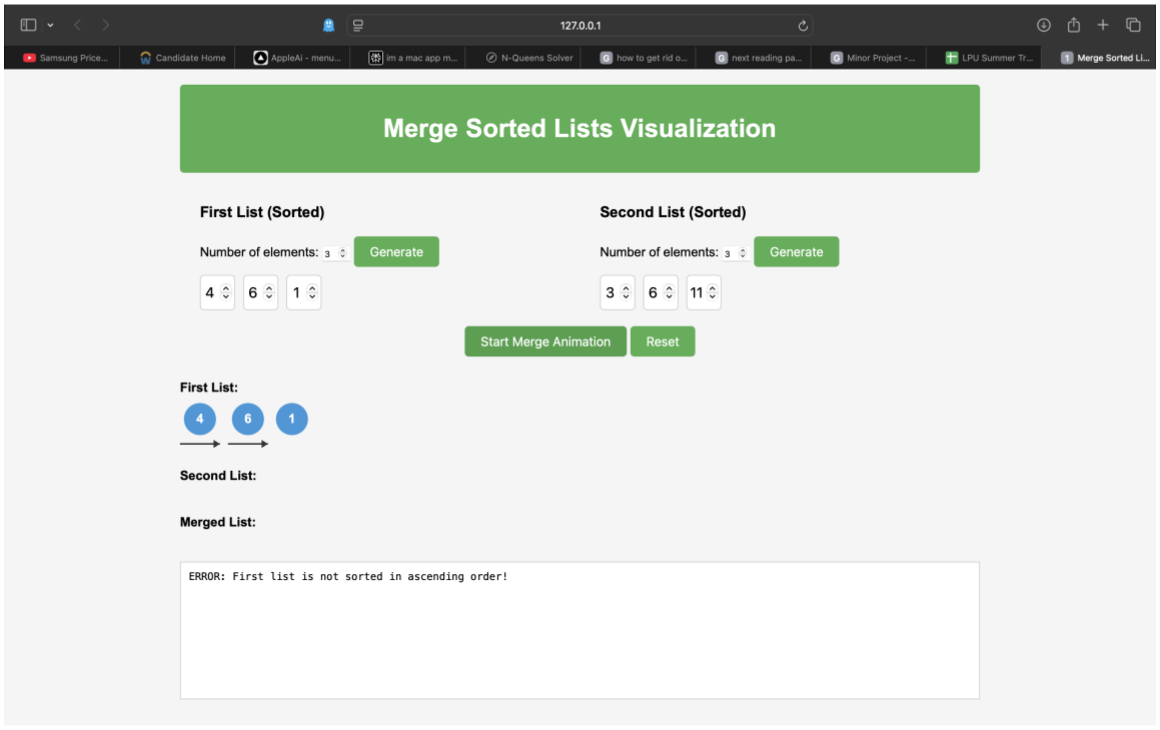
This project implements a solution to merge two sorted linked lists into a single sorted linked list. Linked lists are fundamental data structures in computer science that consist of nodes where each node contains data and a reference to the next node. The merging of two sorted linked lists is a common operation that demonstrates the efficiency of pointer manipulation and the power of linked data structures.

GitHub Repo: https://github.com/mirzasayzz/Merge-Two-Sorted-Linked-Lists.git

Live Demo: https://mirzasayzz.github.io/Merge-Two-Sorted-Linked-Lists/

2. Problem Statement

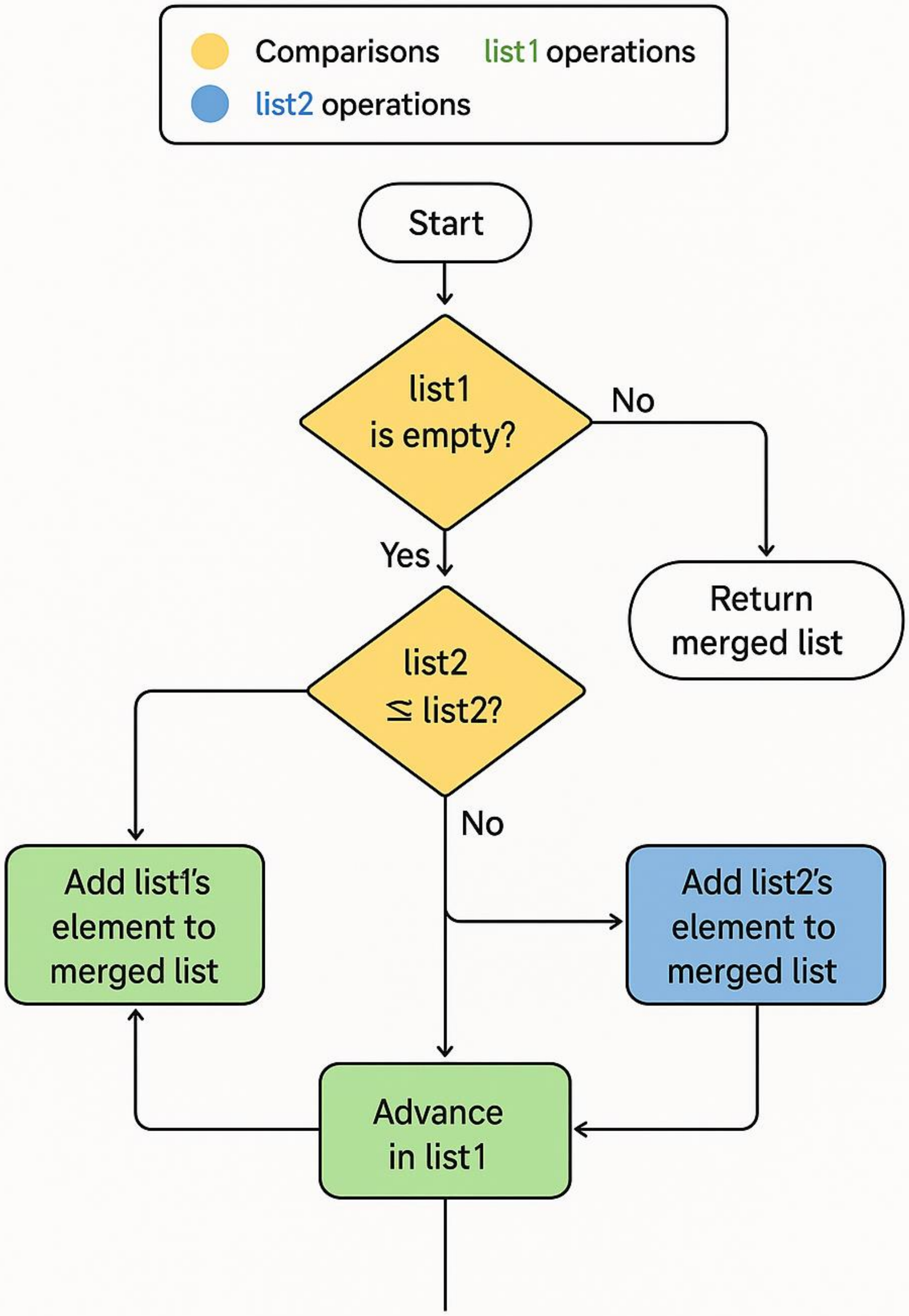
Given two sorted linked lists, we need to merge them into a single sorted linked list. The new list should be made by splicing together the nodes of the first two lists, maintaining their sorted order. The problem tests understanding of linked list manipulation and pointer operations.



```
ERROR: First list is not sorted in ascending order!
```

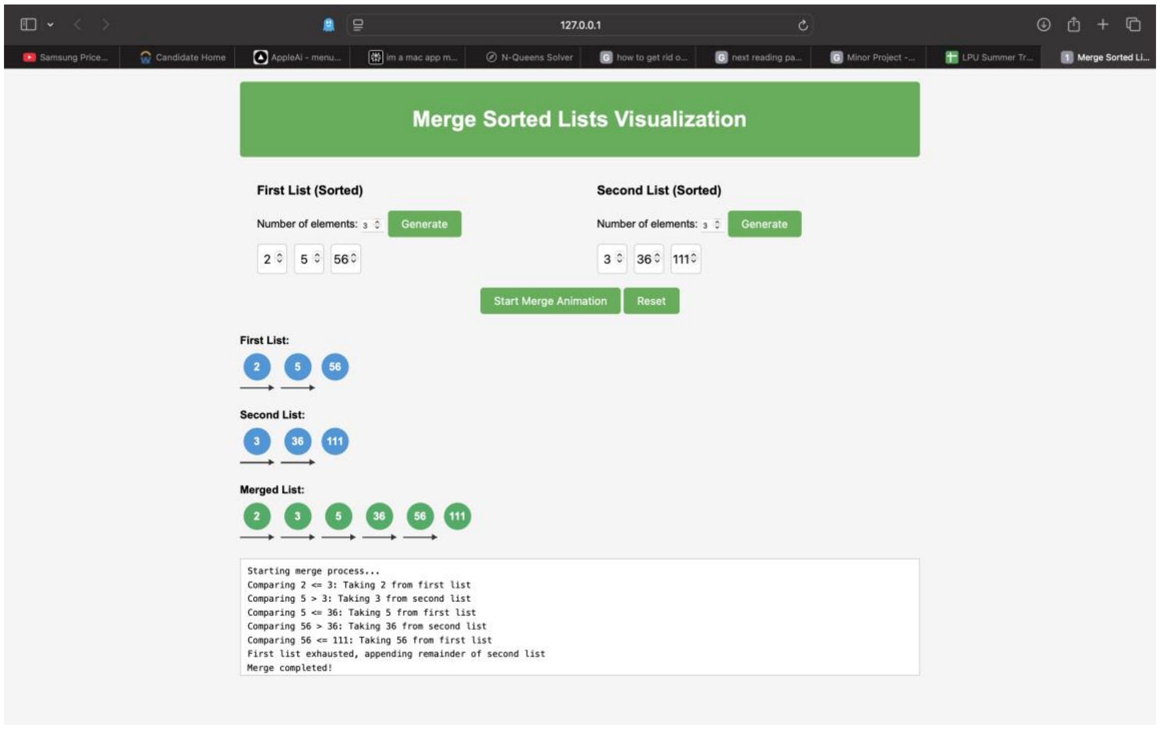
Always Sorted List

Merge Sorted Lists Algorithm Flowchart



Example:

List 1: 1 -> 3 -> 5 -> 7
List 2: 2 -> 4 -> 6 -> 8
Merged List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8



```
Starting merge process...
Comparing 2 <= 3: Taking 2 from first list
Comparing 5 > 3: Taking 3 from second list
Comparing 5 <= 36: Taking 5 from first list
Comparing 56 > 36: Taking 36 from second list
Comparing 56 <= 111: Taking 56 from first list
First list exhausted, appending remainder of second list
Merge completed!
```

Visual Implementation

3. Algorithm Description

The algorithm for merging two sorted linked lists uses a step-by-step approach:

• Create a dummy node as the starting point of the merged list.

• Initialize a current pointer to the dummy node.

• Compare the current nodes of both lists and append the smaller value node to the result list.

• Advance the pointer in the list from which the node was selected.

• Repeat steps 3-4 until one of the lists is fully processed.

• Append any remaining nodes from the non-empty list to the result list.

• Return the merged list (excluding the dummy node).

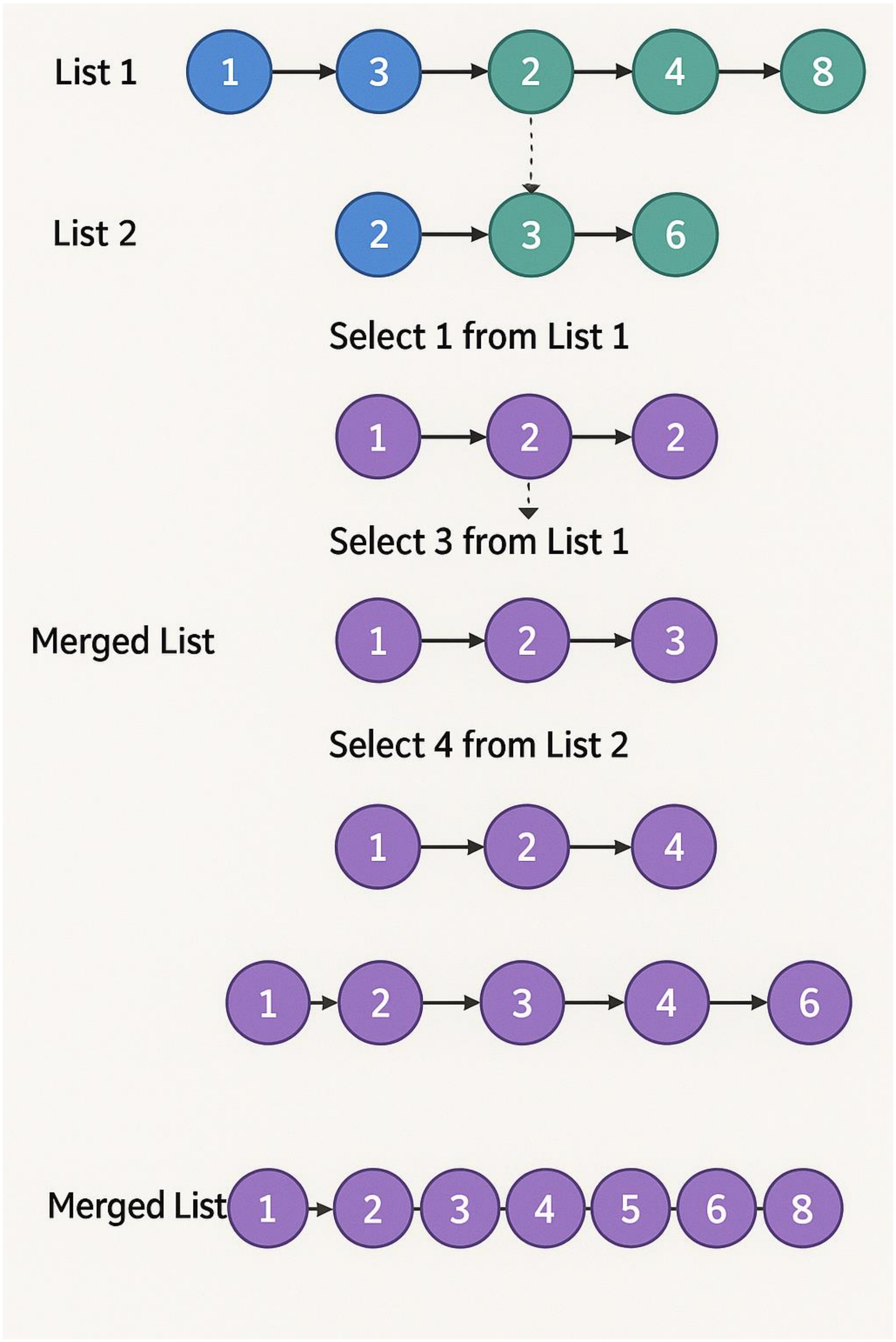
Merge Process Visualization:

Step 1: Initial State
List 1: 1 -> 3 -> 5
List 2: 2 -> 4 -> 6
Merged: Empty

Step 2: Compare 1 vs 2
List 1: 3 -> 5 (1 selected)
List 2: 2 -> 4 -> 6
Merged: 1

Step 3: Compare 3 vs 2
List 1: 3 -> 5
List 2: 4 -> 6 (2 selected)
Merged: 1 -> 2

Final Result:
Merged List: 1 -> 2 -> 3 -> 4 -> 5 -> 6


List 1
1
3
2
4
8
List 2
2
3
6
Select 1 from List 1
1
2
2
Select 3 from List 1
Merged List
1
2
3
Select 4 from List 2
1
2
4
1
2
3
4
6
Merged List
1
2
3
4
5
6
8

4. Implementation

4.1 Data Structures

The implementation uses a Node structure to represent each element in the linked list:

```
struct Node {
    int data;      // Value stored in the node
    Node* next;    // Pointer to the next node

    // Constructor
    Node(int val) {
        data = val;
        next = nullptr;
    }
};
```

4.2 Functions

insertAtEnd: Inserts a new node at the end of a linked list

displayList: Displays the contents of a linked list

mergeSortedLists: Merges two sorted linked lists into a single sorted list

freeList: Frees the memory allocated for a linked list

The key function is the mergeSortedLists function:

```
Node* mergeSortedLists(Node* list1, Node* list2) {
    // If either list is empty, return the other list
    if (list1 == nullptr) return list2;
    if (list2 == nullptr) return list1;

    // Create a dummy node as the head of the result list
    Node* dummyNode = new Node(0);
    // Pointer to track the current position in the result list
    Node* current = dummyNode;

    // Traverse both lists
    while (list1 != nullptr && list2 != nullptr) {
        // Compare current nodes from both lists
        if (list1->data <= list2->data) {
            current->next = list1;
            list1 = list1->next;
        } else {
            current->next = list2;
            list2 = list2->next;
        }
        current = current->next;
    }

    // If one list is not fully traversed, append the remaining nodes
    if (list1 != nullptr) {
        current->next = list1;
    } else {
        current->next = list2;
    }

    // Return the merged list (excluding the dummy node)
    Node* mergedList = dummyNode->next;
    delete dummyNode;
    return mergedList;
}
```

4.3 Main Program

The main program provides a user interface that allows the user to input two sorted linked lists. It then calls the mergeSortedLists function and displays the result.

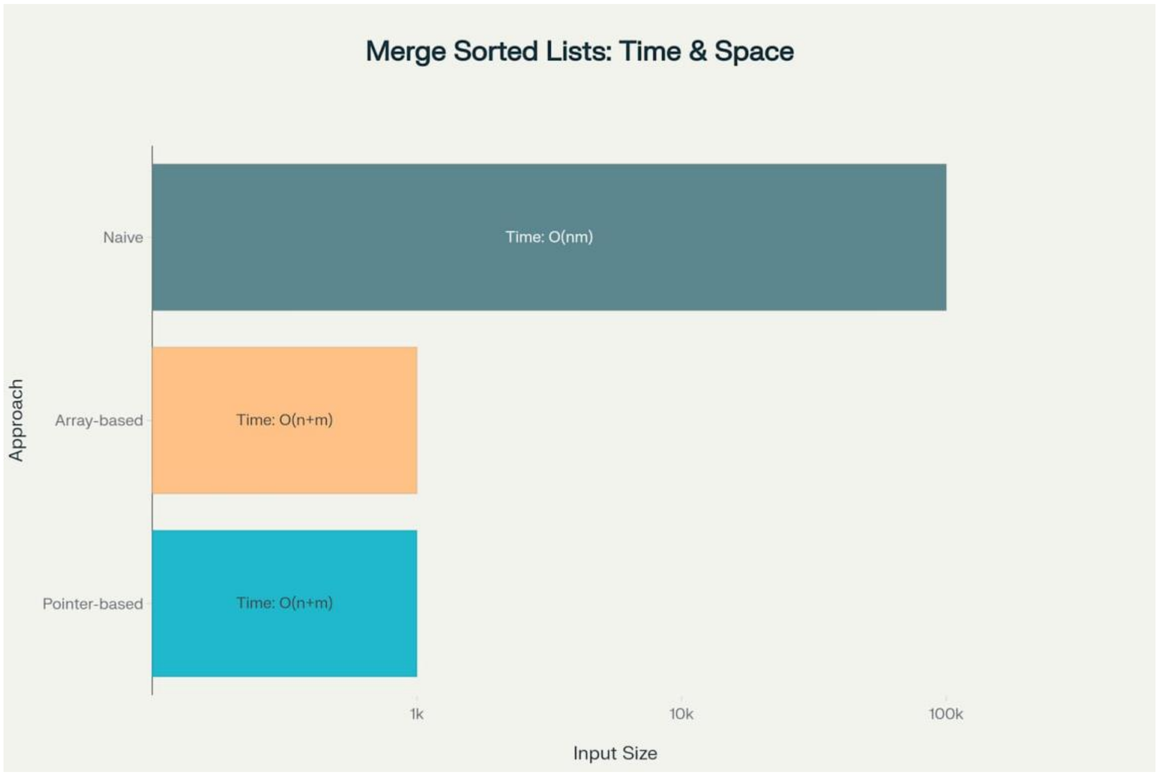
5. Execution and Testing

The program can be executed from the command line after compilation. Below are sample inputs and outputs that demonstrate the functionality of the program:

Program Execution Flow:

1. Start
2. Create empty lists
3. Input list sizes
4. Input list elements
5. Display original lists
6. Merge lists
7. Display merged list
8. Free memory
9. End

6. Time and Space Complexity Analysis



Time Complexity: O(n + m) where n and m are the lengths of the two input lists. This is because we need to traverse both lists once.

Space Complexity: O(1) as we are only using pointers to existing nodes and not creating any new nodes (except the dummy node).

Algorithm Efficiency Comparison:

```
Merge Sorted Lists: O(n + m) - Most efficient
Array Merging:      O(n + m) - Similar efficiency
Naive Approach:     O(n * m) - Less efficient
```

7. Conclusion

This project demonstrates the implementation of an algorithm to merge two sorted linked lists. The solution efficiently combines the two lists by comparing elements and rearranging pointers, without creating new nodes or copying data. This implementation showcases the power and flexibility of linked lists as a data structure.