Task 5 report

1. Introduction

This report provides a functional analysis and complexity assessment of an algorithm designed to manipulate a sequence of characters. The program accepts a numerical input, generates a pattern based on that input, and subsequently rearranges the elements using a two-pointer technique. The final result is a segregated sequence of characters printed to the standard output.

2. Execution Logic

The execution of the program can be divided into four steps: Input & Sizing, Initialization, Rearrangement, and Output.

Phase 1: Input and Sizing

The process begins by accepting a single integer input from the user. The entered integer is then doubled. The new doubled value determines the total capacity for the data structure used in the next steps.

Phase 2: Pattern Initialization

The program allocates a dynamic list (array) of characters with a size equal to the doubled input value. It then iterates through every position in this list to assign initial values based on the index:

- If the index is an even number, the character 'D' is assigned.
- If the index is an odd number, the character 'L' is assigned.

This makes the vector containing an alternating sequence starting with D and ending in L.

Phase 3: The Rearrangement Strategy (Two-Pointer Approach)

The core logic of the program involves rearranging these characters using two pointers technique. One pointer starts at the very beginning of the list and the other starts at the very end of the list

The program enters a cycle that continues as long as the left pointer is positioned before the right pointer. Within each cycle, the following actions occur in order:

1. Swap: The character at the left pointer's position is strictly exchanged with the character at the right pointer's position.
2. Left Scan: The left pointer increments continuously until it lands on a character equal to 'D'.
3. Right Scan: The right pointer decrements continuously until it lands on a character equal to 'L'.

This process effectively swaps elements at the boundaries and then narrows the active range by skipping over characters that are already in a position relative to the sorting logic.

Phase 4: Output

Once the left and right pointers meet or cross, the rearrangement phase terminates. The program then iterates through the entire modified list from beginning to end, printing each character followed by a space to the console.

3. Complexity Analysis

The time complexity of the algorithm is O(N) as the maximum iteration for any loop is the length of the array itself and does not include any nested loops. The same thing goes for the memory as the memory required double the input n.

4. GitHub