# Question 5(b): Algorithm Analysis Report

## Objective

The goal is to analyze and compare the runtime performance of the two stack implementations used in Question 5(a): Linked List–based Stack and Array–based Stack. The focus is on their insertion (push) and deletion (pop) operations, as well as their space and performance characteristics.

## 1. Time Complexity Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Linked List Stack | Array Stack | Explanation |
| Push (Insert) | O(1) | O(1) (amortized) | Both stacks insert at the top in constant time. For arrays, resizing may occasionally increase the cost, but it remains O(1) on average. |
| Pop (Delete) | O(1) | O(1) | Both remove the top element in constant time. |
| Peek (Access Top) | O(1) | O(1) | Direct access to the top element. |
| IsEmpty() | O(1) | O(1) | Checks whether the stack is empty in constant time. |

Conclusion: Both implementations achieve O(1) time complexity for core stack operations.

## 2. Space and Memory Usage

|  |  |  |
| --- | --- | --- |
| Aspect | Linked List Stack | Array Stack |
| Memory Efficiency | Requires extra space for node pointers. | Uses a single contiguous block of memory. |
| Dynamic Sizing | Fully dynamic — can grow or shrink as needed. | Fixed size unless manually resized. |
| Cache Performance | Lower, due to scattered nodes in heap memory. | Higher, due to contiguous memory layout. |
| Memory Allocation Cost | High, due to frequent new and delete calls. | Low, as allocation is done once at creation. |

Conclusion: The array-based stack is more memory-efficient and cache-friendly, while the linked list stack provides better flexibility for dynamic data sizes.

## 3. Performance Comparison

|  |  |  |
| --- | --- | --- |
| Criterion | Linked List Stack | Array Stack |
| Speed (Runtime) | Moderate | Fast |
| Memory Overhead | Higher | Lower |
| Flexibility | Very Flexible | Limited |
| Cache Locality | Poor | Excellent |
| Best For | Dynamic/Unknown-size data | Fixed-size or predictable data |

## 4. Summary of Findings

Both stack implementations have the same asymptotic time complexity (O(1)) for push and pop operations. The Linked List Stack has higher memory overhead due to node pointers and slower performance due to dynamic allocation. The Array Stack offers faster performance because of contiguous memory and better CPU cache utilization, but it requires predefined capacity.

## 5. Conclusion

Use a Linked List–based Stack when the data size changes frequently or is unknown, and flexibility is more important than memory efficiency. Use an Array–based Stack when the maximum stack size is known beforehand, and you want better speed and lower memory overhead. Both structures perform similarly in terms of algorithmic complexity, but the Array Stack is faster and more memory-efficient in practice, while the Linked List Stack is more adaptable for dynamic workloads.