Q1a

Runtime Analysis Report

Operation Linked List Stack Array Stack

Push O(1) — adds node to top using pointer manipulation O(1) — adds element at top index

Pop O(1) — removes top node O(1) — decrements index

Peek O(1) O(1)

Memory Allocation Dynamic (each node allocated separately) Contiguous block allocated once

Space Efficiency Lower (extra pointer in each node) Higher (pure data storage)

Flexibility Expands automatically (no size limit) Fixed capacity (needs resizing)

Performance Slightly slower due to pointer overhead and heap allocation Faster due to direct indexing in contiguous memory

Summary

Both stacks have O(1) push and pop operations.

Linked list stack uses more memory and has slightly more overhead due to dynamic allocation.

Array stack is faster but less flexible; it can overflow if capacity is reached.

For small or fixed-size data (like this assignment), array-based stack performs better.

For unknown or growing data sizes, linked list stack is more reliable.

Q1b

Runtime Analysis Summary

Implementation Push / Pop Time Memory Use Speed Notes

Linked list stack O(1) each Higher (extra pointers) Slightly slower Dynamic growth, no overflow

Array stack O(1) each Lower Faster Fixed capacity (but efficient)

Both iterative versions are O(k) per pattern branch (same as recursion) — stack just replaces the call stack.