Assignment 6:

Medians and Order Statistics & Elementary Data Structures

Anna Levinskaia

ID 005038541

Algorithms and Data Structures (MSCS-532-B01)

University of the Cumberlands

November 21, 2025

Assignment 6: Medians and Order Statistics & Elementary Data Structures

1. Overview

This assignment explores two core areas in algorithm design and data structure implementation. Part 1 focuses on algorithms that select the k-th smallest element in an array—specifically, a deterministic selection algorithm with worst-case linear time and a randomized algorithm with expected linear performance. Part 2 involves implementing elementary data structures (arrays, stacks, queues, and linked lists) and analyzing their performance, time complexity, and real-world applications.

PART 1: Selection Algorithms

1.1 Implementation Summary

Randomized Quickselect – Expected O(n)

Randomized Quickselect is a divide-and-conquer algorithm that chooses a pivot at random, partitions the array into elements less than or greater than the pivot, and recursively selects the part containing the desired element. It is efficient, in-place, and effective with duplicates.

Deterministic Median-of-Medians – Worst-Case O(n)

This algorithm divides the array into groups of five, finds the median of each group, and recursively selects the median of medians as the pivot. It guarantees worst-case O(n) time and predictable behavior across all inputs.

1.2 Time Complexity Analysis

Randomized Quickselect:

- Best Case: O(n)

- Average Case: O(n)

- Worst Case: O(n^2)

- Space Complexity: O(1)

Deterministic Median-of-Medians:

- Worst Case: O(n)

- Average Case: O(n)

- Space Complexity: O(n)

1.3 Empirical Analysis (Example Results)

Input Size (n)	Quickselect	Median-of-Medians
1000	0.0012 s	0.0034 s
5000	0.0069 s	0.0178 s
10000	0.0151 s	0.0394 s
20000	0.0322 s	0.0811 s

Observations:

- Quickselect is faster in practice due to lower overhead.

- Median-of-Medians is slower but stable.

- Both scale linearly with input size.

PART 2: Elementary Data Structures

2.1 Implemented Structures

- Dynamic Array
- Matrix (nested arrays)
- Stack (using array)
- Queue (using array)
- Singly Linked List

2.2 Time Complexity Analysis

Arrays:

- Access: O(1)
- Insert/Delete in middle: O(n)
- Insert at end: O(1) average

Stacks:

- Push, Pop, Peek: O(1)

Queues:

- Enqueue: O(1)
- Dequeue: O(n) with array, O(1) with linked list

Linked Lists:

- Insert at head: O(1)

- Delete by value: O(n)

- Access by index: O(n)

2.3 Trade-Offs

Arrays vs Linked Lists:

- Arrays are best for fast access.

- Linked lists are best for frequent insertions/deletions.

2.4 Practical Applications

Arrays: scientific computing, lookup tables

Stacks: recursion, undo/redo, DFS

Queues: BFS, OS scheduling

Linked Lists: dynamic memory, hash table chaining

3. Conclusion

Part 1 demonstrated the performance and trade-offs of deterministic vs randomized selection algorithms. Part 2 explored how different data structures offer unique advantages depending on operational needs. Understanding these concepts supports efficient, scalable system design.

Data Structure Discussion

Arrays:

Arrays provide fast O(1) random access because elements are stored in contiguous memory. However, insertions and deletions in the middle require shifting elements, resulting in O(n) performance. They are ideal for static datasets and scientific computations.

Stacks:

Stacks operate on the LIFO principle, offering O(1) push, pop, and peek operations. They are widely used in recursion, DFS, backtracking, and undo/redo systems.

Queues:

Queues follow FIFO order. Array-based queues suffer from O(n) dequeue operations due to shifting elements, but linked-list queues perform enqueue and dequeue in O(1). Useful in OS scheduling, BFS, and messaging systems.

Linked Lists:

Linked lists consist of nodes connected by pointers. They support O(1) insertion at the head but O(n) access and search times. Linked lists shine in dynamic memory situations, implementing stacks/queues, and collision handling in hash tables.

Trade-Offs:

- Arrays: Best for fast indexing.

- Stacks: Best for LIFO processes.

- Queues: Best for FIFO order.

- Linked Lists: Best for dynamic and frequent insert/delete operations.

Conclusion:

Each data structure offers unique performance characteristics. Arrays excel at fast access, stacks and queues enforce ordered operations, and linked lists provide memory flexibility. Understanding their trade-offs allows developers to choose the optimal structure for algorithmic efficiency and system performance.

GitHub:

https://github.com/AnnaLevin/MSCS532_Assignments/tree/main/Assignment6