Assignment 6: Medians and Order Statistics & Elementary Data Structures

University of the Cumberlands, Department of Computer Science

Written by: Carter Nelson

Instructor: Vanessa Cooper

08/03/2025

Part 1: Selection Algorithms

1.1 Implementation Overview

There are two algorithms implemented to find the k^{th} smallest element (order statistic):

- **Deterministic Selection (Median of Medians)**: This algorithm guarantees worst-case linear time $O(n)$ by selecting a pivot that is close to the true median. The array is divided into groups of 5, and the median of each group is found. The median of these medians is then used as the pivot.
- **Randomized Selection (Quickselect)**: This algorithm follows the quicksort-style partitioning with a randomly chosen pivot. It has an expected time complexity of $O(n)$ but may degrade to $O(n^2)$ in the worst case.

Both implementations handle edge cases, including duplicate elements and small arrays.

1.2 Time and Space Complexity Analysis

Algorithm	Best Case	Average Case	Worst Case	Space Complexity
Deterministic Select	O(n)	O(n)	O(n)	O(n)
Randomized Quickselect	O(n)	O(n)	O(n^2)	O(log n)

- **Deterministic Select** achieves linear time by ensuring balanced partition sizes through a carefully chosen pivot.
- **Randomized Quickselect** is faster in practice but can degrade with unlucky pivot selections.

1.3 Empirical Analysis

Both algorithms were run on datasets of varying sizes (1,000 to 1,000,000 elements) and distributions (random, sorted, reverse-sorted).

Observations:

- **Randomized Quickselect** outperforms deterministic select on average for all input sizes.
- **Deterministic Select** showed more stable performance, especially on adversarial inputs (sorted or reverse-sorted).
- As input size increased, the performance gap between them narrowed.

Python's `time` module and plotted performance were done with `matplotlib`.

Part 2: Elementary Data Structures

2.1 Implementations

Arrays and Matrices:

- **Operations**: Insertion, deletion, access by index.
- Implemented a basic 2D matrix class with row/column operations.

Stacks and Queues (Array-based):

- **Stack Operations**: `push`, `pop`, `peek`.
- **Queue Operations**: `enqueue`, `dequeue`, `peek`.

Linked Lists:

- **Operations**: `insert_front`, `insert_back`, `delete`, `search`, `traverse`.
- Implemented singly linked list with node class and pointer manipulation.

(Optional) Rooted Trees:

- Represented each tree node using a linked list structure.

2.2 Time Complexity Analysis

Operation	Array	Linked List
Access	O(1)	O(n)
Insertion (end)	O(1)*	O(n)
Insertion (mid)	O(n)	O(n)
Deletion	O(n)	O(n)
Stack Ops	O(1)	O(1)
Queue Ops	O(1)	O(1)

*Python lists are dynamic arrays with amortized O(1) append.

2.3 Practical Applications

- **Arrays**: Ideal for indexed access, used in image processing and matrices.
- **Stacks**: Used in parsing, function calls (call stack), undo operations.
- **Queues**: Used in task scheduling, BFS traversal, simulations.
- **Linked Lists**: Suitable when frequent insertions/deletions are required and memory reallocation is a concern.

Trade-offs:

- Arrays are faster for access but slower for mid-list insertions/deletions.

- Linked lists are memory-efficient and flexible but incur overhead in traversal.

Conclusion

This assignment provided hands-on experience with both theoretical and practical aspects of selection algorithms and fundamental data structures. Through implementation and empirical testing, we observed how algorithm design choices affect performance. Additionally, building data structures from scratch reinforced understanding of their internal mechanics and trade-offs.

Both parts of the assignment highlight the importance of choosing the right data structure or algorithm based on the problem context, data characteristics, and performance constraints.

References

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
- Hoare, C. A. R. (1961). Algorithm 65: Find. *Communications of the ACM*, 4(7), 321–322.
- Blum, M., Floyd, R. W., Pratt, V., Rivest, R. L., & Tarjan, R. E. (1973). Time bounds for selection. *Journal of Computer and System Sciences*, 7(4), 448–461.
- Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching* (2nd ed.). Addison-Wesley.