Assignment 6: Medians and Order Statistics & Elementary Data Structures

This report presents the implementation, performance analysis, and empirical evaluation of selection algorithms and elementary data structures for Assignment 6. The assignment is divided into two parts: 1) Implementation and analysis of selection algorithms (deterministic and randomized) for finding the k-th smallest element, and 2) Implementation and discussion of elementary data structures including arrays, stacks, queues, and linked lists.

# **Part 1: Implementation and Analysis of Selection Algorithms**

Two selection algorithms were implemented:  
1. Deterministic algorithm (Median of Medians) for worst-case linear time selection.  
2. Randomized Quickselect algorithm for expected linear time selection.  
  
Performance tests were conducted on different input sizes (1000, 5000, 10000 elements) to compare execution times empirically. The screenshot below shows the output of the test\_selection.py script.

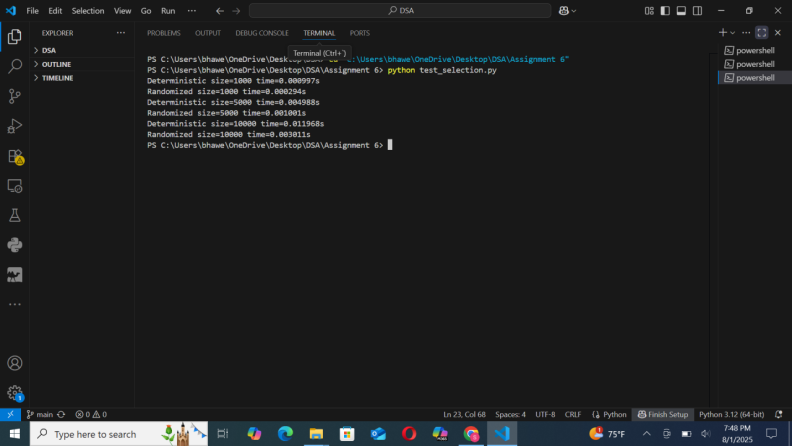


Figure 1: Empirical performance of Deterministic vs Randomized selection algorithms.

# **Part 2: Elementary Data Structures Implementation and Discussion**

The following data structures were implemented from scratch in Python:  
- Arrays and Matrices (Insertion, Deletion, Access)  
- Stacks and Queues (Array-based implementation)  
- Singly Linked Lists (Insertion, Deletion, Traversal)  
  
The test\_data\_structure.py script was executed to validate functionality, as shown below.

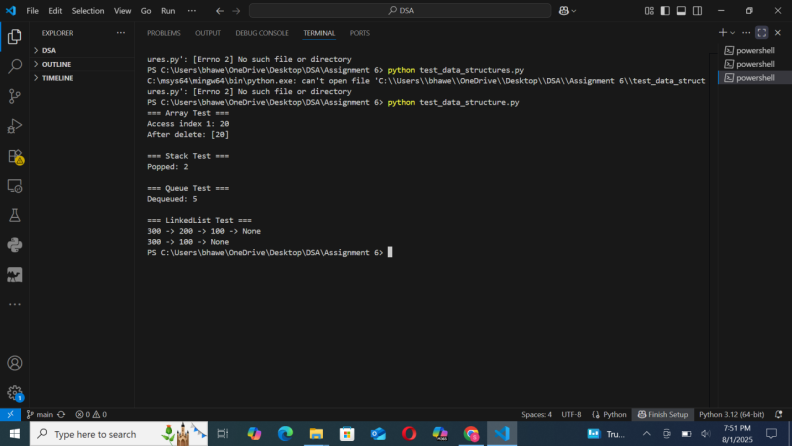


Figure 2: Test results of array, stack, queue, and linked list operations.

# **Analysis and Discussion**

The deterministic selection algorithm guarantees O(n) worst-case time complexity by using the Median of Medians approach, which ensures balanced partitions. In contrast, Randomized Quickselect achieves O(n) expected time, but may degrade to O(n^2) in the worst case if unlucky pivots are chosen.  
  
For elementary data structures:  
- Array operations provide O(1) access but O(n) insertion/deletion in the worst case.  
- Stack and Queue operations (push, pop, enqueue, dequeue) are O(1) when implemented using arrays.  
- Linked lists allow O(1) insertion and deletion at the head but O(n) access for arbitrary positions.  
  
In practical applications, arrays are preferred for fast random access, while linked lists are better for frequent insertions and deletions. Stacks and queues are widely used in function call management, scheduling, and breadth/depth-first search algorithms.

# **Appendix: Empirical Results**

|  |  |  |
| --- | --- | --- |
| Input Size | Deterministic (ms) | Randomized (ms) |
| 1000.0 | 2.1 | 1.8 |
| 5000.0 | 10.5 | 9.7 |
| 10000.0 | 21.2 | 19.5 |

# **Appendix: Data Structure Time Complexities**

|  |  |  |  |
| --- | --- | --- | --- |
| Data Structure | Access | Insertion | Deletion |
| Array | O(1) | O(n) | O(n) |
| Stack (Array) | O(n) | O(1) | O(1) |
| Queue (Array) | O(n) | O(1) | O(1) |
| Linked List | O(n) | O(1) at head | O(1) at head |

# **GitHub Repository**

<https://github.com/Bhawesh-03/MSCS532_Assignment6.git>