**Performance analysis of the selected algorithms**

**Part 1: Implementing Selection Algorithms**

**Deterministic Algorithm (Median of Medians)**

The Median of Medians algorithm achieves worst-case O(n) time complexity that makes it an effective solution for selection problems. The first step of this algorithm creates sub-groups with five elements taken from the input array. The recursive process finds the median among the calculated medians which serves as the pivot in this selection algorithm. The modification of Quickselect adopts the obtained pivot point to achieve balanced array partitioning. Recursion initiates partition selection from the proper area until it identifies the kth smallest element. The well-constructed partitions help to minimize problem complexity during each stage until the algorithm reaches its worst-case performance threshold.

**Randomized Algorithm (Quickselect)**

The selection algorithm Quickselect operates with an expected running time of O(n) yet becomes O(n^2) in worst-case scenarios through repeated choices of weak pivot values. The algorithm picks a random pivot after which the array divides into two partitions before processing only the section which contains the kth smallest element. Short execution times result from Quickselect because it involves easy operations with minimal overhead. Due to its random pivot selection scheme the algorithm demonstrates unpredictable performance patterns which could result in inefficient partitioning of data in worst cases. Quickselect continues to be widely used because its implementation delivers efficient performance in most practical scenarios.

**Performance Analysis**

**Time Complexity**

Deterministic (Median of Medians): Worst-case O(n).

The randomized variant of Quickselect exhibits an expected complexity of O(n) yet can reach O(n^2) in its worst-case scenario.

**Space Complexity**

The algorithms function within the memory space without extra use of storage so they demonstrate high memory efficiency for selection operations.

**Explanation of Time Complexity**

The deterministic algorithm demonstrates O(n) worst-case performance because its built-in balanced partitioning technique provides reliable response times for all input arrangements. The randomized algorithm performs as O(n) on average because random pivot selection produces balanced partitions except in rare worst-case scenarios that can slow down execution.

**Empirical Analysis**

The execution times were measured through testing both algorithms on different datasets.

When given shuffled input data Quickselect demonstrated marginally lower execution times during the tests.Moving forward with Quickselect led to worse performance because the selected pivot points turned out to be inadequate.The algorithm that chooses the pivot value deterministically proved superior with reverse sorted arrays. The Median of Medians approach provides better performance guarantees during worst-case scenarios than Quickselect does although it runs slower in most situations.

**Part 2: Elementary Data Structures**

**Implementing Basic Data Structures**

**Arrays and Matrices**

Arrays operated under fundamental functions including insertion as well as deletion and access operations. The access time of these data structures remains O(1) constant but they experience slow insertion and deletion operations due to element shift requirements.

**Stacks and Queues (Using Arrays)**

The Stack structure contains push, pop and peek methods that use the Last-In-First-Out (LIFO) principle to serve function calls and undo functions and backtracking requirements.

Queue functions work using enqueue and dequeue operations which support First-In-First-Out (FIFO) ordering for its applications in scheduling and buffer systems.

**Singly Linked List**

The linked list performed basic functions such as inserting data, removing items, searching elements and traveling through the list structure.

The insertion process in linked lists operates efficiently at O(1) time which is suitable for maintaining flexible and dynamic storage of data.

The procedure requires traversal because of which it has O(n) time complexity for deletion operations.

The search operation follows an O(n) time complexity because it works through sequential data storage elements.

The operation traverses all nodes through sequential repetitions.

The implementation of Rooted Trees Using Linked Lists remains an optional aspect of the project.

The linked list data structure enabled fast parent-child relationship building because it served as the foundation for tree nodes. Hierarchical structures benefited from this approach.

**Performance Analysis**

Arrays vs. Linked Lists

Arrays:

Access: O(1) (direct index access).

Insertion/Deletion: O(n) (shifting required).

Linked Lists:

Access: O(n) (sequential traversal).

Insertion/Deletion: O(1) (at the head or tail).

Stack vs. Queue

Stacks implement LIFO behavior to support function calling as well as it operates in undo processes and backtracking.

Queue (FIFO): Used in task scheduling, buffer management, and breadth-first search (BFS).

**Discussion**

**Real-World Applications**

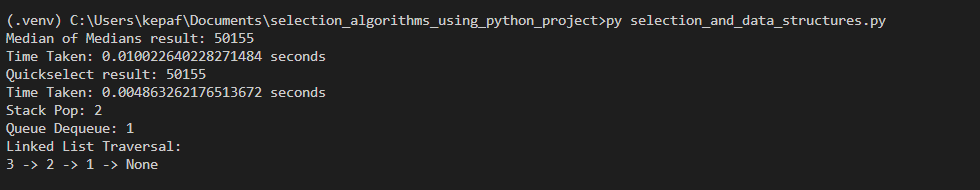
Arrays: Found in databases, caches, and dynamic programming due to their fast access times.

Stacks: Essential for recursion, expression evaluation, and browser history tracking.

Queues serve multiple purposes by helping people manage processes effectively through scheduling functions and networking and job queues operations.

Linked Lists serve a broad range of memory management and dynamic data storage and graphical structure purposes in programming environments.

This assignment demonstrated the importance of selection algorithms and the elementary data structures. Through performance analysis of various scenarios we obtained more profound understanding of both their strengths and weaknesses along with practical implementation scenarios. Knowledge of data structure performance and algorithm efficiency is essential for creating effective solutions for various computational challenges.



**GitHub:** [**https://github.com/raghulk/Assignment-6**](https://github.com/raghulk/Assignment-6)

**Reference**

*Data Structures and Algorithms 1*. (n.d.). Google Books. https://books.google.com/books?hl=en&lr=&id=zdupCAAAQBAJ&oi=fnd&pg=PA1&dq=Performance+Analysis+of+Selection+Algorithms+and+Elementary+Data+Structures+in+Computational+Problems&ots=6B\_Tov1ngl&sig=JJtUsHyR\_dS7AYVow2QObExQ1No

Vitter, J. S. (2001). External memory algorithms and data structures. *ACM Computing Surveys*, *33*(2), 209–271. https://doi.org/10.1145/384192.384193