**TOPIC:**

**STUDENT NAME:**

**UNIVERSITY NAME:**

**COURSE NAME:**

**INSTRUCTOR NAME:**

**DATE OF SUBMISSION:**

**PART 1:**

Performance Analysis:

Time Complexity

Deterministic Algorithm (Median of Medians): O(n)

Randomized Algorithm (Randomized Quickselect): O(n)

Space Complexity

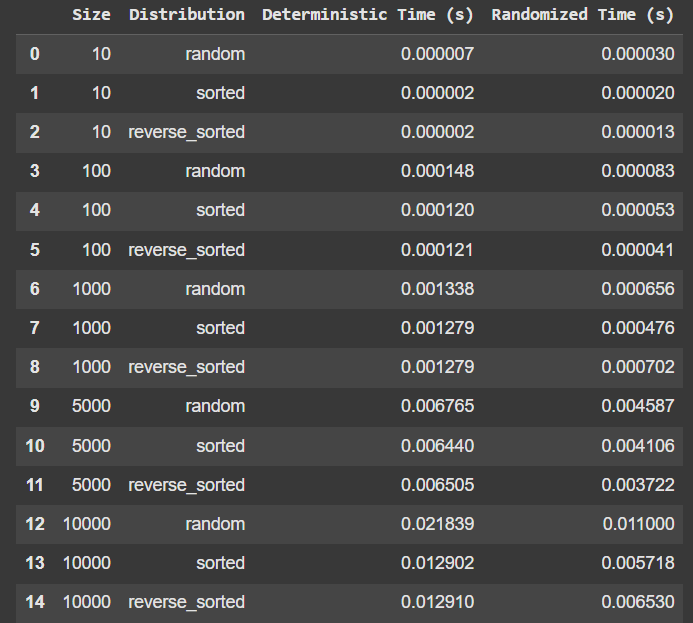
The space complexity for both the algorithms are O(log n).

Comparison and Key Insights

|  |  |  |
| --- | --- | --- |
| Aspect | Deterministic Algorithm | Randomized Algorithm |
| Worst-Case Time | O(n) | O(n²) |
| Expected Time | O(n) | O(n) |
| Space Complexity | O(log n) | O(log n) |
| Robustness | Guarantees performance | Performance may degrade in rare cases |
| Ease of Implementation | Relatively complex | Simpler |

Empirical Analysis:

This is the output:



General Performance Trends:

Deterministic Algorithm:

Execution time grows nearly linearly with the input size because of the dependency on the precise partitioning of the data (Median of Medians). However, the increase is linear as the expected time complexity from the ‘For’ loop symbol O(n).

Randomized Algorithm:

The speed is higher compared to the deterministic algorithm in all sizes and distributions due to the application of random pivoting. Time complexity remains at par with the usual O(n) depending on the given scenarios.

Impact of Input Distribution:

Both algorithms perform well with sorted and reverse sorted inputs with only slight increases in runtime compared to random inputs.

This further validates the effectiveness of the partitioning strategies holding for any order of input.

Randomized Algorithm:

Some differences in the time taken are because of the random selection of the pivot.

Scaling with Input Size:

Both algorithms demonstrate approximately linear scaling:

For n=10: Even the time of execution of both the algorithms is almost insignificant.

For n=10,000: The average time of deterministic is approximately two times slower than randomized (21.839ms vs. 11.000ms for random input).

Deterministic vs. Randomized:

Small Inputs (n=10):

A randomized algorithm is slower because additional time is required for choosing a random pivot for small arrays.

Medium and Large Inputs (n≥100):

Since randomized has a lower constant overhead it can be seen that it is consistently better than deterministic.

**PART 2:**

Performance analysis:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation | Array (Dynamic) | Stack (Array) | Queue (Array) | Linked List (Singly) |
| Insert | O(n)\* | O(1) | O(1) | O(1) |
| Delete | O(n)\* | O(1) | O(1) | O(n)\*\* |
| Access | O(1) | O(1) | O(1) | O(n) |
| Traversal | O(n) | O(n) | O(n) | O(n) |

Trade-Offs: Arrays vs. Linked Lists:

Memory Usage:

Arrays occupy a continuous place and hence may not be efficient for large data or for where we are likely to change the size frequently.

The linked lists allow memory allocation on a dynamic basis which saves space but adds the cost of the pointers.

Insertion/Deletion:

It has been also important to know that linked lists enable O(1) time on insertion/deletion from the head /tail while in arrays, it may mean shifting elements with the complexity of O(n).

Arrays are used full in direct indexing and random access than linked lists.

Stack/Queue Implementation:

Arrays make stack/queue simpler since all stack/queue operations on the end/start have O(1).

Linked lists do not resize but they entail pointer management.

Comparison of Data Structures:

Scenarios:

High-frequency insertions/deletions:

Linked lists should be used when we need to insert or delete an element in O(1) time and without shifting.

Random access:

Linked list is worse, as elements in them must be traversed.

Memory-constrained environments:

Linked lists do not grow out of the box but cost more per node.