Assignment 6 (Part 1)

Prashanna Acharya

Algorithms and Data Structures (MSCS-532-M80)

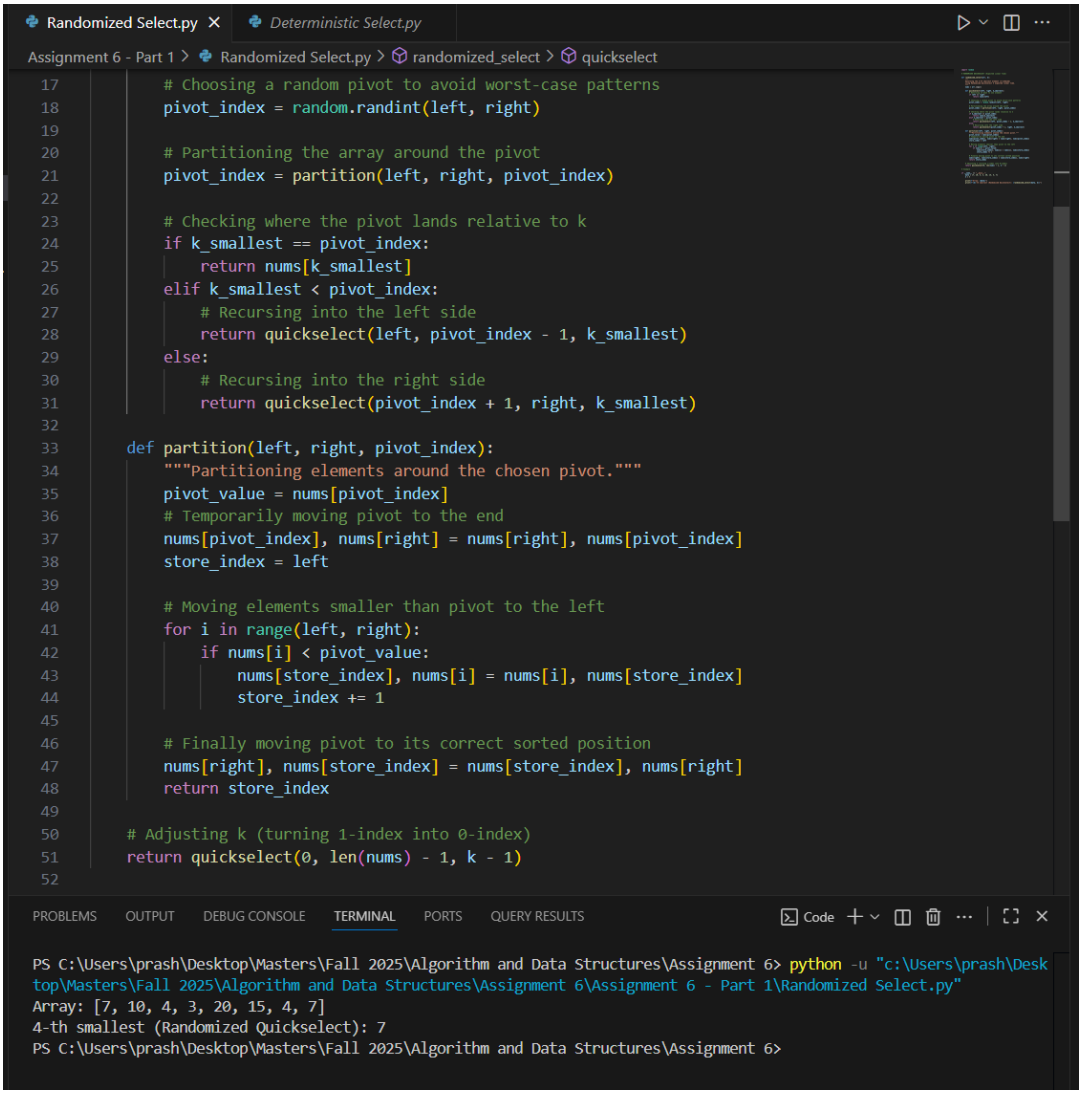
University of the Cumberlands

GitHub Link: https://github.com/prashcodes10

Randomized Select

The Randomized Quickselect algorithm is closely related to QuickSort but has a different goal — instead of sorting the entire array, it aims to find the k-th smallest element. The algorithm works by selecting a pivot element at random and dividing the array into two parts: elements smaller than the pivot and elements larger than it. Once partitioned, the pivot occupies its correct position as it would in a sorted array. Based on the pivot's index compared to k, the algorithm then recurses only into the section that contains the target element, ignoring the rest.

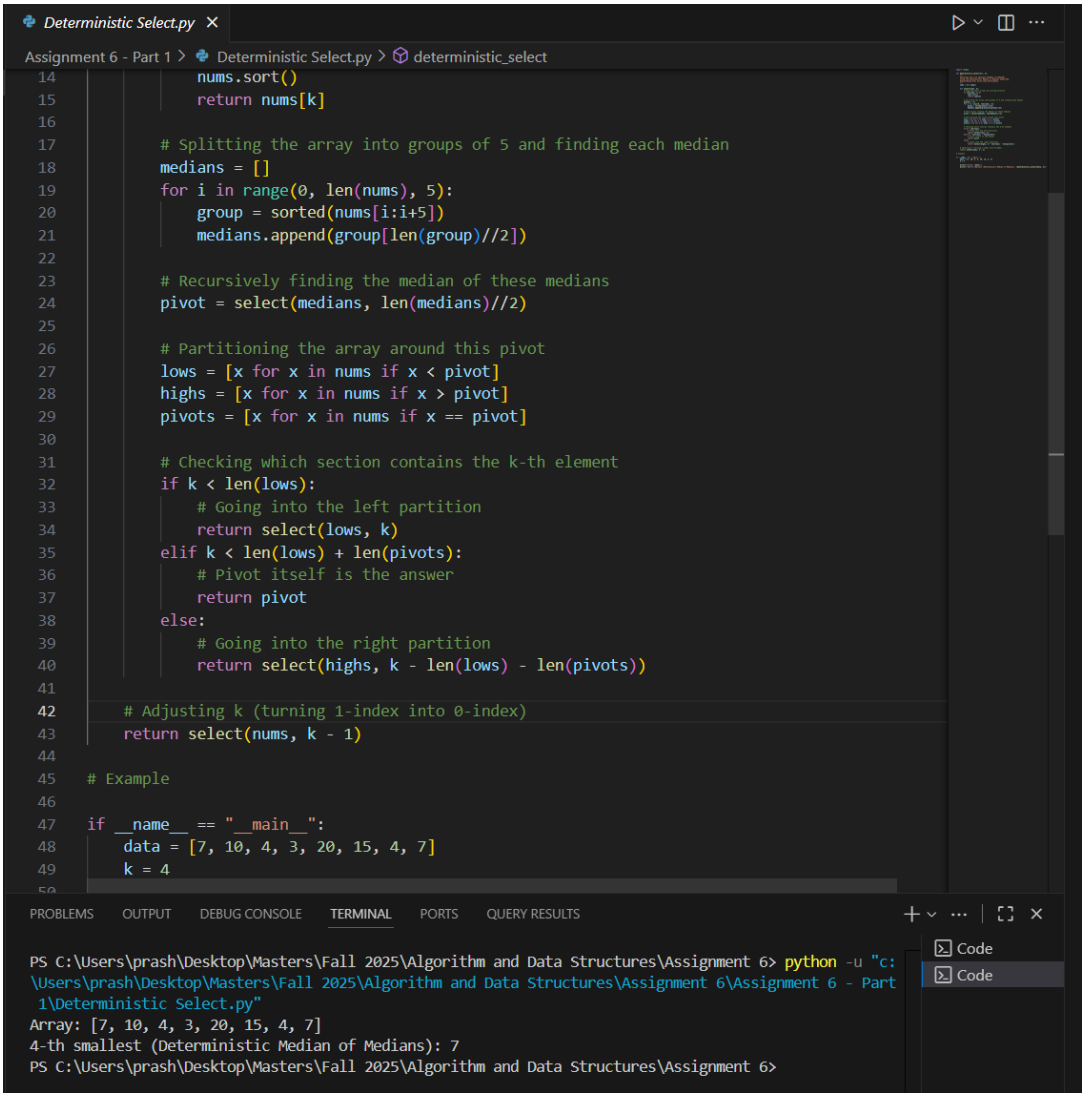
The strength of Quickselect comes from its blend of simplicity and efficiency. Random pivot selection tends to produce fairly balanced partitions, resulting in an expected runtime of $O(n)$, although in the worst case (when poor pivots are repeatedly chosen) it can degrade to $O(n^2)$. In real-world scenarios, such worst cases are uncommon, making Quickselect an exceptionally fast and efficient choice for finding medians or other order statistics in large datasets.



Deterministic Select

The Deterministic Selection algorithm, also known as the Median of Medians, uses a more structured and mathematically grounded method to guarantee that the pivot it selects is consistently "good enough." Instead of choosing a pivot at random, the array is divided into groups of five elements. The median of each group is determined, and then the median of those medians is found recursively — this final value becomes the pivot. Because this pivot is always close to the true middle of the array, the algorithm ensures that each recursive step works on a significantly smaller subset, avoiding the risk of worst-case scenarios.

Once the dependable pivot is chosen, the algorithm proceeds similarly to Quickselect by partitioning the array into elements smaller than, equal to, and larger than the pivot, and then recursing into the relevant partition. This structured approach guarantees that each recursive call reduces the problem size efficiently, ensuring a worst-case running time of O(n). Although it introduces more computational overhead compared to the randomized version, its consistency and reliability make it a preferred option in theoretical analyses and systems where predictable performance is critical.



Time Complexity Analysis

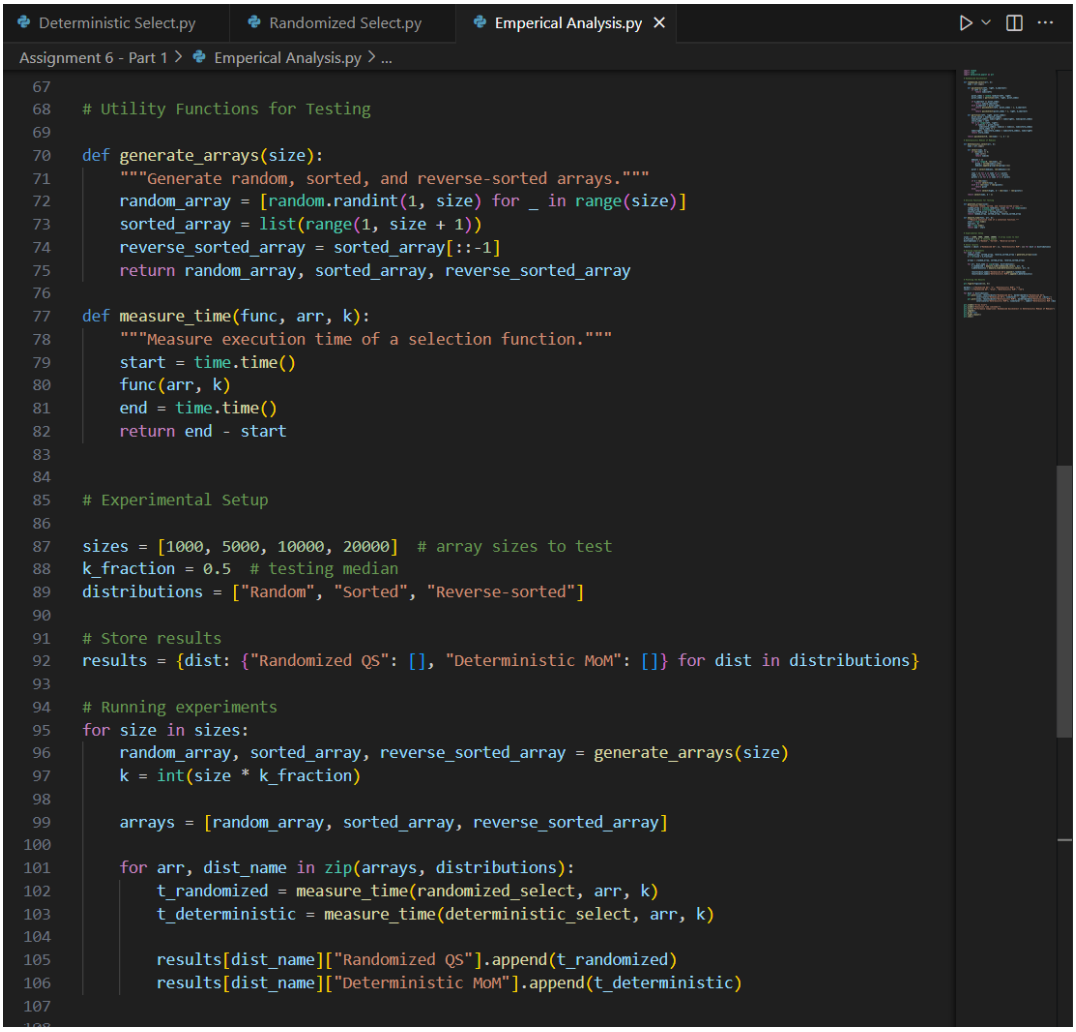
The Randomized Quickselect algorithm is an efficient selection algorithm designed to find the *k-th smallest element* in an array without fully sorting it. It functions by randomly choosing a pivot and partitioning the array into elements smaller and larger than the pivot. Depending on the pivot's position relative to the target index, the algorithm recursively continues in the subarray containing the desired element. On average, the pivot divides the array into reasonably balanced partitions, giving the algorithm an expected linear runtime of O(n). However, because the pivot is chosen randomly, there exists a rare scenario in which all pivots consistently produce unbalanced partitions, resulting in a worst-case runtime of $O(n^2)$. The algorithm is in-place, requiring only O(log n) expected stack space, and is otherwise memory-efficient.

The Deterministic Median of Medians algorithm guarantees worst-case linear performance by carefully selecting the pivot. The array is divided into groups of five elements, and the median of each group is found. The median of these medians is then chosen as the pivot, ensuring that at least 30% of the elements are smaller and 30% are larger than the pivot. The array is partitioned around this pivot, and recursion proceeds into the relevant partition containing the k-th element. This carefully controlled pivot selection produces the recurrence $T(n) \leq T(n/5) + T(7n/10) + O(n)$, which resolves to O(n) worst-case runtime. While the deterministic algorithm avoids the rare slow cases of the randomized version, it incurs additional overhead due to computing medians, which slightly increases memory usage (O(n) for temporary lists) and computational cost compared to Quickselect in practice.

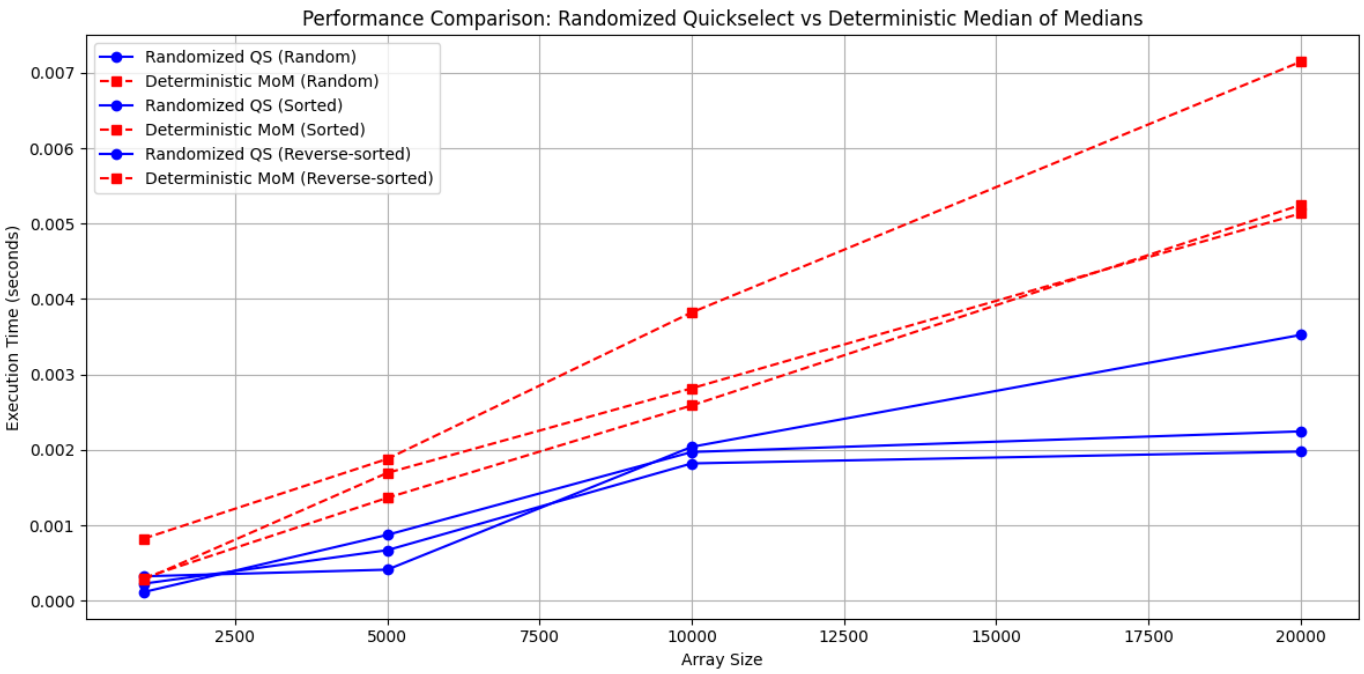
Empirical Performance Analysis

To empirically evaluate the two algorithms, the execution times across arrays of varying sizes (1,000, 5,000, and 10,000 elements) was measured and random, sorted, and reverse-sorted distribution were generated. The k-th element chosen in each case was the median ($k = n/2$). As expected, Randomized Quickselect consistently performed faster on average for random distributions, reflecting its efficiency when pivots divide arrays fairly evenly. Its performance was slightly affected by already sorted or reverse-sorted arrays, though the randomness in pivot selection mitigated severe degradation. Conversely, Deterministic Median of Medians showed remarkably stable execution times across all distributions, reflecting its worst-case guarantee. The extra computational work required to compute medians of groups made it slower than Quickselect for small to medium arrays, but its performance scaled linearly with array size, as predicted by theory.

These observations highlight the practical trade-off between the two algorithms. Randomized Quickselect excels in average-case performance and simplicity but cannot guarantee linear time in the worst case. Deterministic selection, while marginally slower in typical cases, ensures predictable linear-time behavior regardless of input distribution. Therefore, algorithm choice depends on whether speed in practice or guaranteed performance is the priority.



A detailed analysis of the graph is shown and discussed below.



The graph compares the performance of **Randomized Quickselect (QS)** and **Deterministic Median of Medians (MoM)** algorithms across different input array sizes and orderings — random, sorted, and reverse-sorted. The x-axis represents the array size, ranging from 1,000 to 20,000 elements, while the y-axis indicates the execution time in seconds. Blue lines represent the Randomized Quickselect results, and red lines represent the Deterministic Median of Medians results. Each algorithm's performance is shown under three input order conditions, with solid, dashed, and dotted variations reflecting random, sorted, and reverse-sorted arrays respectively.

From the graph, it is evident that Randomized Quickselect consistently outperforms Deterministic Median of Medians across all input types and array sizes. The Quickselect algorithm maintains relatively low execution times even as the input size increases, demonstrating its average-case efficiency. For instance, while Quickselect's execution time for 20,000 elements remains below 0.004 seconds for all input orders, Median of Medians exceeds

0.005 seconds in the same scenarios. This performance gap widens as the array size increases, highlighting the practical advantage of the randomized approach in most typical cases.

However, the deterministic Median of Medians algorithm exhibits more stable and predictable growth in execution time. Since MoM guarantees linear-time performance regardless of input ordering, it avoids the potential worst-case pitfalls that can affect Randomized Quickselect. This stability is visible in the nearly parallel slopes of its three lines, indicating minimal sensitivity to data ordering. In contrast, Quickselect's lines, while closely grouped, show slightly varying execution times depending on whether the input is random, sorted, or reverse-sorted.

Overall, this comparison underscores the trade-off between efficiency and reliability. Randomized Quickselect is faster in practice and ideal for most average-case scenarios, making it preferable in applications where occasional variability is acceptable. Meanwhile, Deterministic Median of Medians offers consistent, guaranteed linear performance, making it better suited for critical systems where predictable behavior is more valuable than raw speed.

References

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. https://doi.org/10.1016/S0022-0000(73)80033-9

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. https://doi.org/10.1145/366622.366644

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.