Unique Karanjit

Department of Computer Science, University of the Cumberlands

MSCS-532-A01: Algorithms and Data Structure

Dr. Satish Penmatsa

Feb 9, 2025

Table of Contents

Deterministic Selection Sort

Introduction:

Deterministic Selection Sort is a selection-based sorting algorithm to choose the k^{th} smallest/largest element in a list without sorting the whole array. One of the most well-known selection-based algorithms is Selection Sort. The process followed is - in each step, select the minimum from the unsorted portion of the list and exchange it with the leading element in the sorted portion.

In the case of Deterministic Selection Sort, we want to find a particular order element (e.g., the median, or the k^{th} smallest) using a deterministic approach; this is typically done with the median of medians algorithm which provides better worst-case time complexity.

Time Complexity:

- Worst-case time complexity: $O(n)$
- Best-case time complexity: $O(n)$
- Average-case time complexity: $O(n)$

Why $O(n)$ in the worst case:

The deterministic Selection Sort works in linear time in the worst case: instead of sorting the whole array, it does this by recursive dividing lists into smaller ones, choosing their medians, and using these as a pivot for partitioning recursively. In the median of medians application, a point is guaranteed to be close to the middle. The step-by-step breakdown is shown below.

1. Divide the array into groups: First, the array is divided into small groups, each containing no more than 5 elements.
2. Find the median of each group: Then for each group, we find the median.

3. Find the median of the medians: The medians of the subgroups are used to recursively find a "good" pivot (median of medians).
4. Partition the array: The array is partitioned around this median and the process is repeated for the part of the array containing the desired k^{th} element.

Since median of medians assures a good pivot, each step of recursion reduces the problem size substantially hence the time complexity of the entire algorithm being $O(n)$

Space Complexity:

- Space complexity: $O(n)$

Space complexity will involve the storage of subarrays, but additional space might also be required for the recursive call stack. In general, the deterministic selection sort algorithm is in place, with linear space usage to maintain the partitioned subsets.

Randomized Selection Sort

Introduction:

Randomized Selection Sort is a variant of the deterministic selection sort wherein the selection of the pivot is random instead of deterministic. This randomness in most cases avoids worst-case performance while still finding the desired k^{th} element efficiently.

The algorithm works similarly to Quickselect, which is a selection algorithm to find the k^{th} smallest element from an unordered list and is a variant of QuickSort.

Time Complexity:

- Worst-case time complexity: $O(n^2)$
- Average-case time complexity: $O(n)$

- Best-case time complexity: $O(n)$

Why $O(n)$ on average but $O(n^2)$ in the worst case:

1. Randomized pivot selection: By randomly selecting the pivot, one averages over the partition sizes; hence, this prevents the worst-case time complexities that might have resulted from repeatedly picking poor pivots.
2. Partitioning: After selecting the pivot, the array is partitioned into elements less than the pivot, the pivot itself, and elements greater than the pivot. Then, the algorithm recurses in the portion of the array that contains the k^{th} element.

Average case:

The pivot will divide the array into roughly equal parts. This will yield each partitioning step taking $O(n)$time. Because the algorithm always recurses on one side of the array, the number of recursions is proportional to $O(n)$.

Worst case:

The worst case happens when the random pivot happens to be the smallest or largest element and recursion always operates on an unbalanced partition with almost the whole array each time. This would yield $O(n^2)$ time complexity.

Space Complexity:

- Space complexity: $O(1)$ excluding the recursion stack.

Randomized Selection Sort is an in-place algorithm since it doesn't require extra space for the second array. Nevertheless, the recursion stack will use space proportional to the depth of recursion which in the worst case is $O(n)$.

Comparative Analysis of Deterministic and Randomized Selection Sort:

Criteria	Deterministic Selection Sort	Randomized Selection Sort
Worst-Case Time Complexity	$O(n)$	$O(n^2)$
Average-Case Time Complexity	$O(n)$	$O(n)$
Best-Case Time Complexity	$O(n)$	$O(n)$
Space Complexity	$O(n)$	$O(1)$
Pivot Selection	Median of Median	Random
Recursion Depth	Limited by partitioning with median of medians	Can be deep in the worst case
Efficiency	$O(n)$	$O(n)$ in best/average, $O(n^2)$ worst

Key Differences:

Worst-case Performance:

- Deterministic Selection Sort ensures $O(n)$ time complexity in the worst case by selecting a good pivot, which does not lead to the worst-case partitioning.
- Randomized Selection Sort can degrade to $O(n^2)$in the worst case if the pivot selection results in highly unbalanced partitions.

Space Complexity:

- Deterministic Selection Sort uses $O(n)$ space because it divides the list into smaller subarrays.

- Randomized Selection Sort does an in-place sorting, hence uses $O(1)$ for the partitioning part and some more stack space for recursion.

Selection of Pivot:

- Deterministic Selection Sort follows the rule of selecting the median of medians as a good pivot, thus avoiding the pitfalls of bad ones.
- Randomized Selection Sort selects pivots randomly; though this generally works well on average, the worst case remains at $O(n^2)$.

Conclusion:

- Deterministic Selection Sort has $O(n)$ worst-case time complexity** because it uses a good strategy for choosing a pivot: the median of medians. This is more reliable in terms of time complexity but with higher space complexity.
- Randomized Selection Sort has an expected time complexity $O(n)$ with the average case being quite efficient. However, its worst-case time complexity could degrade to $O(n^2)$ if the pivots chosen are poorly distributed. It is more space-efficient but less predictable in terms of time performance.

In practice, randomized selection sort is preferred since it is simpler, and worst cases are extremely unlikely to occur for random pivot selection. For situations that demand a guaranteed performance, deterministic selection sort is more reliable.

Empirical Analysis

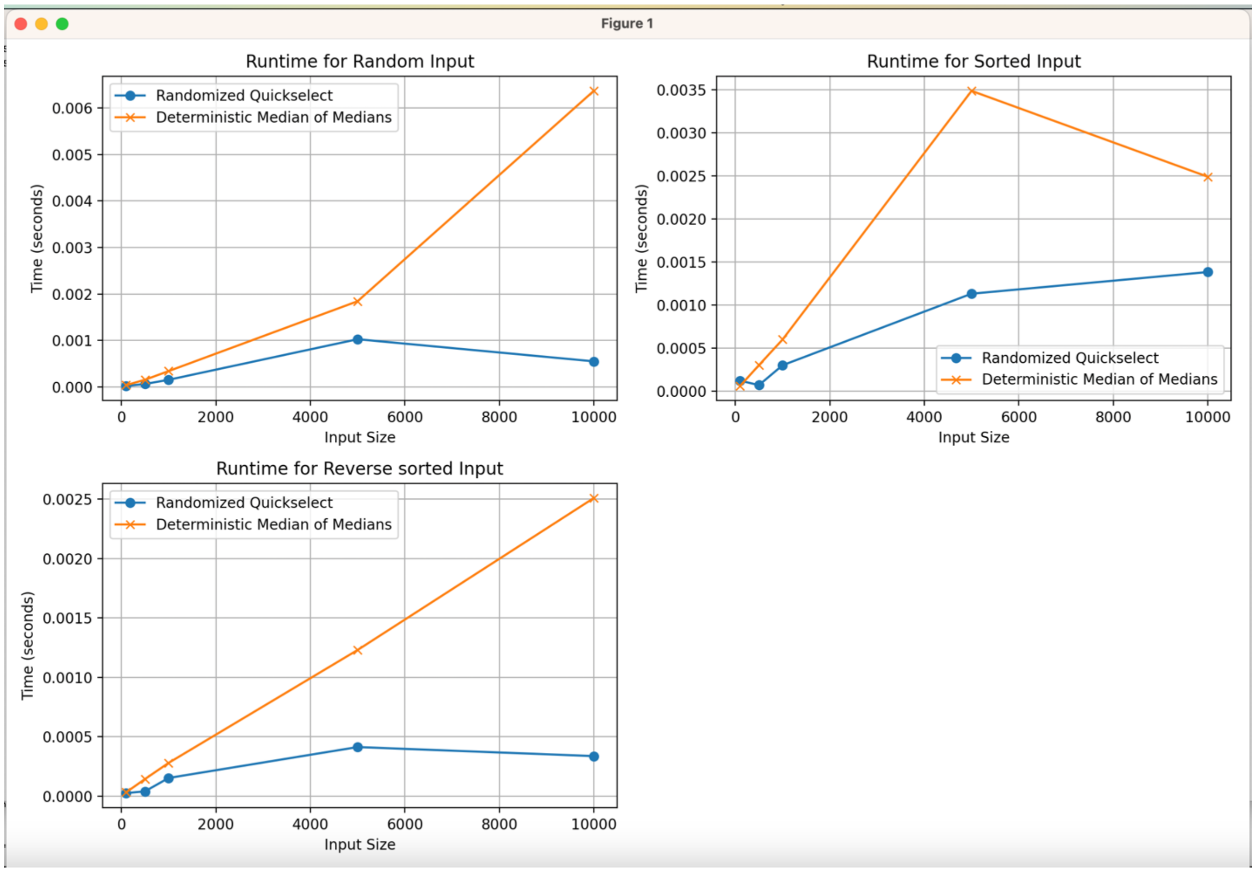
Why is Deterministic Median of Medians Slower than Randomized Quickselect?

1. Overhead of Finding the "Good" Pivot:

- Randomized Quickselect just selects a random pivot in O(1) time.
- Median of Medians first:
    - Divides the array into chunks of 5.
    - Computes the median in each chunk, which itself involves sorting small chunks.
    - Recursively selects the median of these medians as the pivot.
- This extra work blows up its constant factors in its runtime.

2. Randomized Quickselect is Optimized for Practical Cases
    - Picking the pivot randomly works in most cases, which keeps the recursion well balanced.
    - Though Quickselect worst case is O(n^2) - it just doesn't occur in practice.
    - The expected number of recursive calls is much smaller than that of Median of Medians.
3. Median of Medians Has a Higher Constant Factor

- Its worst-case guarantee O(n) comes with some extra work for Median of Medians.
- The constant factor in the O(c * n) for Median of Medians is higher than in Quickselect.
- Quickselect has a small constant factor and, thus is in practice faster most of the time.



Based on the experiment, even if the randomized quickselect has the worst time complexity of $O(n^2)$, in reality, it outperforms the deterministic median of medians since it's almost not possible to hit the worst case.

References

GeeksforGeeks. (2020, March 18). *Selection algorithms*. https://www.geeksforgeeks.org/selection-algorithms

ICS 161: Design and analysis of algorithms lecture notes for January 30, 1996. Deterministic selection. (n.d.). https://ics.uci.edu/~eppstein/161/960130.html

Selection (deterministic & randomized): Finding the median ... (n.d.). https://www.cs.cmu.edu/~avrim/451f12/lectures/lect0906.pdf