Project 2

Chang Li

GitHub: https://github.com/chl0817/Project-2/tree/main

1 Problem Statement

In this project, I am going to analyze the QuickSelect algorithm using the deterministic median of medians method. QuickSelect is an algorithm for finding the k-th smallest element in an unsorted list. The deterministic version ensures a worst-case linear time complexity by selecting an optimal pivot using the median of medians method.

Algorithm: QuickSelect with Median of Medians:

A screenshot of a computer program

Description automatically generated

2 Theoretical Analysis  
The QuickSelect algorithm with the median of medians method has a worst-case time complexity of O(n). By dividing the array into groups of five and selecting the median of medians as the pivot, the algorithm ensures that at least a constant fraction of elements is discarded in each recursive call. This effective pivot selection leads to balanced partitions, reducing the problem size significantly at each step. Consequently, the algorithm achieves linear performance by minimizing the number of recursive calls required to find the k-th smallest element.

3 Experimental Analysis

3.1 Program Listing

Below is the Python code used to conduct the experimental analysis of the QuickSelect algorithm with the median of medians method. The code measures the execution time of the algorithm for various input sizes and plots the results using Jupyter Notebook.

A screenshot of a computer program

Description automatically generated

3.2 Data Normalization Notes

To compare the experimental results with the theoretical linear time complexity, we normalized the execution times by calculating the time per element in microseconds. Since the execution times are in milliseconds, we divided each execution time by the corresponding

n to get the time per element in milliseconds, then multiplied by 1,000 to convert it to microseconds per element. This normalization allows us to observe whether the time per element remains approximately constant across different input sizes, indicating linear scalability.

3.3 Output Numerical Data

Below is the table of the experimental results:

|  |  |  |
| --- | --- | --- |
| n | Experimental Time(ms) | Normalized Time |
| 1000 | 0.2611 | 0.2611 |
| 100000 | 23.9450 | 0.2395 |
| 200000 | 47.0692 | 0.2353 |
| 300000 | 70.6414 | 0.2355 |
| 400000 | 93.3721 | 0.2334 |
| 500000 | 116.2842 | 0.2326 |
| 600000 | 140.9817 | 0.2350 |
| 700000 | 212.6533 | 0.3038 |
| 800000 | 190.2085 | 0.2378 |
| 900000 | 225.9621 | 0.2511 |
| 1000000 | 247.2601 | 0.2473 |

3.4 Graph

A graph with a line

Description automatically generated

A graph with a line going up

Description automatically generated

3.5 Graph Observations

The first graph demonstrates that the execution time increases approximately linearly with the input size n, which aligns with the theoretical O(n) time complexity. The second graph shows that the normalized time per element remains relatively constant around 0.23 to 0.26 microseconds per element for most values of n. There is a noticeable spike at n=700,000, where the normalized time increases to approximately 0.30 microseconds per element. This deviation may be due to system-related factors such as CPU scheduling, memory caching effects, or background processes that affected the performance during that specific measurement.

4 Conclusions

The experimental results confirm the theoretical prediction that the QuickSelect algorithm with the median of medians method operates in linear time, O(n). The execution times increase proportionally with the input size, and the normalized time per element remains consistent across different values of n. Minor deviations observed in the normalized times are likely attributed to external system factors rather than the algorithm's efficiency. Overall, the median of medians approach effectively ensures worst-case linear time complexity, validating its practicality and efficiency for selecting the k-th smallest element in large, unsorted datasets.