Analysis of Algorithms

BLG 335E

Project 1 Report

RENGİN HELİN YALÇIN
yalcinr22@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 21.11.2023

1. Implementation

1.1. Implementation of QuickSort with Different Pivoting Strategies

- The main function of this algorithm is the partition function, which performs the step of rearranging elements in the array. The last element strategy selects the last element as the pivot, the random element strategy randomly selects a pivot, and the median of three strategy chooses the median of three randomly selected elements as the pivot. Additionally, the code makes use of a function, $getRandomPivot$, to generate a random index within a specified range, employing random pivot selection.

- The recurrence relation for the worst-case time complexity of quicksort is often expressed as:

$$T(n) = T(n_1) + T(n_2) + O(n)$$

where:

n_1 and n_2 are the sizes of the two partitions obtained by the partitioning step. $O(n)$ represents the time spent on partitioning.

In the average case, with good pivot selection strategies, the recurrence relation often involves a term like:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

representing the recursive division of the array into two halves.

- In the average case, when the pivot selection is good and the array is balanced, the time complexity is $O(n \log n)$. The space complexity is primarily determined by the recursive calls and the space required for the call stack. In the worst case, the space complexity is $O(\log n)$. In conclusion, the space complexity of the quicksort implementation is $O(\log n)$ in the worst case, where n is the size of array.

	Population1	**Population2**	**Population3**	**Population4**
Last Element	4750100 ns	217677400 ns	182977100 ns	3308200 ns
Random Element	3078600 ns	2270400 ns	2084500 ns	3945700 ns
Median of 3	2724700 ns	1869400 ns	2860500 ns	3587300 ns

Table 1.1: Comparison of different pivoting strategies on input data.

- The last element pivot strategy shows varying performance across datasets, with significant increases in runtime for certain datasets. Whereas, the random element pivot strategy and median element strategy show consistent performance in general.
- The significant increase in runtime for "population2.csv" and "population3.csv" with the last element pivot suggests that these datasets may have characteristics that make the last element pivot less favorable

Conclusion: The random and median element pivot strategies show more consistent and competitive performance across different datasets. This aligns with the expected behavior, as these strategies are designed to reduce the risk of worst-case scenarios.

1.2. Hybrid Implementation of Quicksort and Insertion Sort

- The implemented C++ program employs the quicksort and insertion sort algorithms for sorting a dataset from a CSV file that is stored in a vector pair. The program decides which algorithm to employ based on the provided threshold. The user can decide what pivoting strategy to be used for quicksort. User input is provided as command-line arguments.
- The recurrence relation of the algorithm relies on quicksort, therefore the same recurrence relation points mentioned in the first section apply.
- In the average case, quicksort has a time complexity of $O(n \log n)$. In the worst case, quicksort can degrade to $O(n^2)$. However, with good pivot selection strategies (e.g., median-of-three), the worst-case time complexity is often avoided. The time complexity of insertion sort is $O(n^2)$ in the worst case and $O(n)$ in the best case. Insertion sort is used when the size of the data is less than or equal to the threshold k. Considering that quicksort is applied when the size of the data is greater than the threshold k, the overall time complexity of the program is influenced by the behavior of quicksort. In practice, with good pivot selection strategies, the average time complexity is often close to $O(n \log n)$.

Threshold (k)	10	600	1000	10.000	100.000
Population4	3913000 ns	4280200 ns	3946200 ns	4898800 ns	1293732500 ns

Table 1.2: Comparison of different thresholds on the hybrid QuickSort algorithm on input data.

For very large datasets (e.g., when the threshold is 100,000), the runtime increases significantly. This is very likely due to the fact that insertion sort becomes less efficient for large subarrays.

The runtime does not consistently increase or decrease with each increase in the threshold. Since I used random pivoting for this experiment, this fluctuation might be influenced by various factors like the randomness introduced by the choice of pivot.