# **Assignment 2: Analyzing and Implementing Divide-and-Conquer Algorithms**

Jacob Jeffers

MSCS 532

Dr. Vanessa Cooper

Fall 2024

11/3/2024

Assignment 2: Analyzing and Implementing Divide-and-Conquer Algorithms

Part 1: Asymptotic Analysis and Recurrence Relations

For Part 1, I chose to analyze the Quick Sort and Merge Sort algorithms. The Quick Sort algorithm uses the divide-and-conquer method to sort an array *n* in non-decreasing order (Cormen et al., 2022, p.188). It involves selecting an element from the array (usually the last element). From there, it rearranges the elements so that all elements less than the selected number come before, and all elements greater come after. The same procedure is applied to the subarrays (if necessary). For time complexity, the best-case scenario is Ω(n log (n)). This suggests that if the partitioning is equally balanced at every level of recursion, an asymptotically faster algorithm results (Cormen et al., 2022, p.188). The worst-case scenario is O(n^2). This suggests that the worst-case scenario for this algorithm is that it is no faster than the insertion sort algorithm (Corman et al., 2022). The average case's time complexity analysis would be Θ(n log(n)). This would be the average case if the split is constantly proportional at each recursion level (Corman et al., 2022).

**Recurrence Relation**

**Average Case**: T(n)=2T(n/2)+O(n)

**Substitution Method**

T(n)=2T(n/2)+O(n)

T(n)=2T(n/2​)+cn

T(n)=2(2T(n/4)+c(n/2)​)+cn=4T(n/4​)+2cn

T(n)=4(2T(n/8​)+c(n/4)​)+2cn=8T(n/8​)+3cn

T(n)=2^kT(n/(2^k)​)+k⋅cn

T(n)=2^log2​n⋅T(1)+(log2​n)⋅cn

T(n)=Θ(nlogn)

Quick Sort is often faster than Merge Sort due to lower constant factors and fewer memory requirements but can be slower in the worst-case scenario (Cormen et al., 2022).

The Merge Sort algorithm attempts to solve a similar problem to the Quick Sort algorithm – it seeks to take an array of numbers and sort them in non-decreasing order (Cormen et al., 2022). Essentially, this algorithm splits an array into two halves, sorts each half, and then combines the sorted halves to produce the final array. The time complexity analysis for the best-case scenario is Ω(nlogn), the worst-case scenario is O(nlogn), and the average-case scenario is Θ(nlogn) (Cormen et al., 2022).

**Recurrence Relation**

T(n)=2T(n/2)+O(n

**Substitution Method**

Expanding the recurrence, we find that each level requires O(n) time, and there are log(n) levels, resulting in T(n)=Θ(n log(n)).

Merge Sort is helpful because it is stable and works well with larger datasets, but it is less efficient for certain sort types when compared to Quick Sort (Cormen et al., 2022).

Part 2: Implementation and Comparison

The Quick Sort example I created was fundamental. Below is the screenshot for the input and output of the code run in VS Code.

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

The algorithm chooses a pivot (a middle element number) and partitions elements to the less than, equal to, and more significant than the chosen number. Then, the quick sort algorithm is called and concatenates the results. The code was executed in 0.058 seconds.

A black screen with white text

Description automatically generated

For the Merge Sort algorithm, the code created in VS Code was easier to do than Quick Sort. I think I may better understand the Merge Sort algorithm or the coding practice for the first, which made the second one easier. The code looks like the following.

A screen shot of a computer program

Description automatically generated

The code executed the same array in 0.098 seconds.

A black screen with white text

Description automatically generated

This suggests that the Quick Sort algorithm is roughly two times faster. However, more data is needed to determine this, and the array will change. After playing with the array size, I discovered that the Merge Sort algorithm is faster when the array is already mostly sorted.

For example, if I use the following array [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], the merge sort algorithm sorts this in 0.078 seconds, while the quick sort algorithm sorts it in 0.087 seconds.

A black rectangular object with brown lines

Description automatically generated

Analysis: For Quick Sort algorithms, poor pivot choices can degrade performance on sorted or reverse-sorted arrays. Merge Sort appears to be more consistent.

GitHub Repository for Assignment 2: <https://github.com/jakejeffers/MSCE-532-Assignment-2>

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms, fourth edition*. MIT Press.