COMP 363 - Algorithms: Homework #1

Due on 22 January, 2026

Leo Irakliotis

Nathan Hogg

Problem 1

Background

The goal of this assignment is to prove the time complexity of the mergesort algorithm. We are given the base complexity formula as:

$$T(n) = 2T(n/2) + f(n)$$

In this case, $T(n)$ is the time it takes to mergesort an array of size n, and $f(n)$ is the time it takes to assemble two partial solutions of size $n/2$ to a sorted array with n elements. Generally, $f(n) \approx n$.
For an array with $n = 8$, we can write:

$$T(8) = 2T(4) + f(8)$$

Then, $T(4) = 2T(2) + f(4)$; and $T(2) = 2T(1) + f(2)$; and finally $T(1) = f(1)$. Using the last finding, we can solve backwards:

$$\begin{aligned} T(2) &= 2T(1) + f(2) \\ &= 2f(1) + f(2) \end{aligned}$$

$$\begin{aligned} T(4) &= 2T(2) + f(4) \\ &= 2(2f(1) + f(2)) + f(4) \\ &= 4f(1) + 2f(2) + f(4) \end{aligned}$$

$$\begin{aligned} T(8) &= 2T(4) + f(8) \\ &= 2(4f(1) + 2f(2) + f(4)) + f(8) \\ &= 8f(1) + 4f(2) + 2f(4) + f(8) \end{aligned}$$

Given that $f(n) \equiv n$, the last equation can be rewritten as:

$$\begin{aligned} T(8) &\equiv 8 \times 1 + 4 \times 2 + 2 \times 4 + 8 \\ &= 32 = 8 \times 3 \\ &= 8(1 + \log_2 8) \end{aligned}$$

We have shown that, for mergesort, $T(n) \equiv n \log_2 n$ (or, $\mathcal{O}(n \log_2 n)$),

Proof

We begin by expanding the recurrence relation:

$$\begin{aligned} T(n) &= 2T\Big(\frac{n}{2}\Big) + f(n) \\ &= 2\left[2T\Big(\frac{n}{4}\Big) + f\Big(\frac{n}{2}\Big)\right] + f(n) \\ &= 2\left[2\left[2T\Big(\frac{n}{8}\Big) + f\Big(\frac{n}{4}\Big)\right] + f\Big(\frac{n}{2}\Big)\right] + f(n) \\ &= 2\left[2\left[2\left[2T\Big(\frac{n}{16}\Big) + f\Big(\frac{n}{8}\Big)\right] + f\Big(\frac{n}{4}\Big)\right] + f\Big(\frac{n}{2}\Big)\right] + f(n) \end{aligned}$$

By distributing the outer coefficients into the brackets, we see a pattern emerge:

$$T(n) = 16T\Big(\frac{n}{16}\Big) + 8f\Big(\frac{n}{8}\Big) + 4f\Big(\frac{n}{4}\Big) + 2f\Big(\frac{n}{2}\Big) + f(n)$$

Problem 1 continued on next page...

We observe that for each term, the coefficient matches the denominator inside the function. Since the cost function for Mergesort is $f(n) = n$, we can substitute this into each term of the sum (e.g., $8 \cdot f(n/8) = 8 \cdot (n/8) = n$):

$$\begin{aligned} T(n) &= 16T\Big(\frac{n}{16}\Big) + n + n + n + n \\ &= 16T\Big(\frac{n}{16}\Big) + 4n \end{aligned}$$

We can see that for a recursion of depth k, the first term becomes $2^k T(n/2^k)$, and we add n exactly k times. This can be generalized to the following:

$$T(n) = 2^k T\Big(\frac{n}{2^k}\Big) + k \cdot n$$

The recursion stops when the array size becomes 1 (the base case). If we set the array size $\frac{n}{2^k} = 1$, it implies that $2^k = n$. Solving for k:

$$\frac{n}{2^k} = 1 \implies 2^k = n \implies k = \log_2(n)$$

Finally, we substitute $k = \log_2(n)$ and $2^k = n$ back into the general formula derived above:

$$T(n) = n \cdot T(1) + n \log_2(n)$$

Conclusion

We have successfully derived the expression for the recurrence relation. The final formula is:

$$T(n) = n \times T(1) + n \log_2(n)$$

If we assume that the base case, $T(1)$, takes a constant amount of time, denoted as c, then the equation becomes:

$$T(n) = c \times n + n \log_2(n)$$

As n approaches infinity, the term $n \log_2(n)$ grows much faster than the linear term $c \times n$. Therefore, the linear term becomes negligible, and we conclude that the time complexity of Mergesort is:

$$T(n) \in \mathcal{O}(n \log n)$$