Recurrences

One of the most important tools for analyzing algorithms involves determining and then solving recurrences. A recurrence is an equation or inequality that describes a function in terms of its value on smaller and smaller inputs. Recurrences are typically used to characterize the performance of recursive functions/algorithms. As such, they are commonly applied in the analysis of "divide-and-conquer" algorithms. We considered the MergeSort algorithm in the previous unit as an example of a divide-and-conquer algorithm. In that unit, we characterized MergeSort's complexity using the recurrence:

$$\begin{aligned} T(n) &= \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases} \\ &= \Theta(n \lg n). \end{aligned}$$

The question facing us is how we determined that the solution to this recurrence was $\Theta(n \lg n)$? More importantly, what tools are available to us to solve any recurrence?

In the following, we will discuss four different methods for solving recurrences:

- **The substitution method** – Here we will guess a bound and use mathematical induction to prove whether or not the guess is correct.
- **The iteration method** – For this method, we will convert the recurrence into a summation and bound the summation to solve.
- **Recursion trees** – In this case, we construct a tree showing the reduction of the input space and analyze the size of the tree.
- **The master method** – This approach provides a "cookbook recipe" for solving certain types of recurrences. Specifically, we will consider three cases of recurrences of the form $T(n) = aT(n/b) + f(n)$ where $a \geq 1$, $b > 1$, and $f(n)$ is some arbitrary function.