1 | Motivation

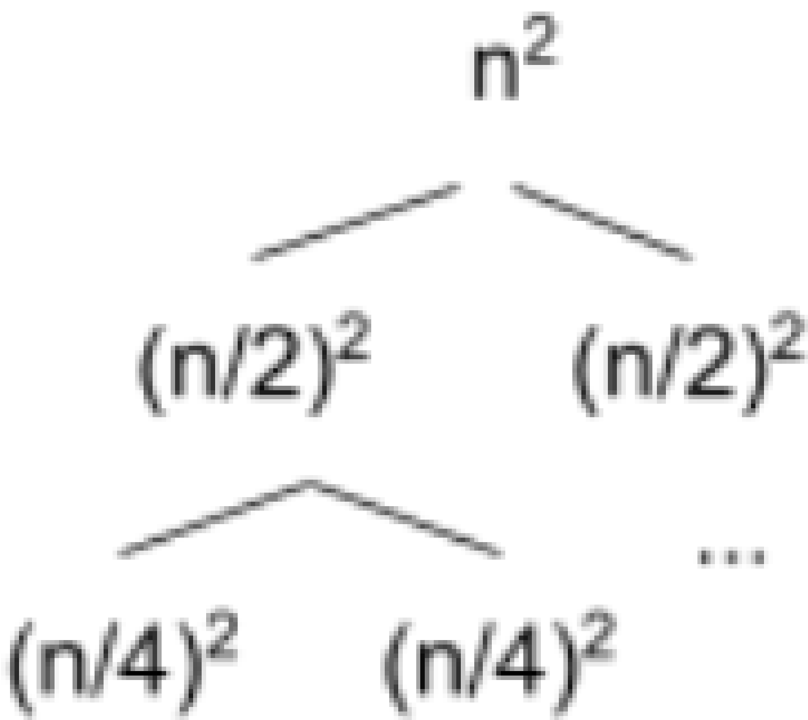
Let's look at the recurrence relation for mergesort:

$$T(n) = 2T(\frac{n}{2}) + \theta(n) \tag{1}$$

We can say a few things:

- $\frac{n}{2}$ is the size of a sub problem
- 2 is the number of sub problems
- $\theta(n)$ is the time it takes to combine

What if, instead of n, we took $\theta(n^2)$ to do the combining steps?
Our tree changes.



- It will take $\theta(n^2)$ operations to combine the top two
- It will take $\theta(\frac{n^2}{2})$ operations to combine the second level
- $\theta(\frac{n^2}{4})$ operations for the third

And eventually, this adds up:

$$(1 + \frac{1}{2} + \frac{1}{4} + \cdots)\theta(n^2) \tag{2}$$
$$= 2\theta(n^2) \tag{3}$$
$$= \theta(n^2) \tag{4}$$

So the actual change of subproblems are not really covered well in terms of elements shifting. How do we then analyze something with different subproblems vs operations?

2 | Master Method

General recurrence form:

$$T(1) = c \tag{5}$$
$$T(n) = aT\left(\frac{n}{b}\right) + \theta(n^d) \tag{6}$$

Where, $a \geq 1, b \geq 2, c > 0, d \geq 0$ are constants. Also, $n = b^k$ for some positive k. The runtime, therefore, is:

$$T(n) = \begin{cases} \theta(n^d), & a < b^d \\ \theta(n^d log(n)), & a = b^d \\ \theta(n^{log_b a}), & a > b^d \end{cases} \tag{7}$$

Voodoo witchcraft! Just plug and chug. Proof is left to Sheldon Axler.