The Master Theorem, Explained

The master theorem is actually just a simple formula that we can use to find the runtime of recursive algorithms. Lets look at an example.

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)^d$$

So what's happening here is that we're looking at the effect of a recursive algorithm on the runtime of a given problem, because it's not always trivial enough to ignore

you can use a, b, and d as the parameters to simply plug into a piecewise function.

where n is the size of the problem, and a, b, and c are simply constants of the problem.

if there is no $f(n)$ then you simply take d to be zero.

$$\begin{cases} O(n^d) & log_b(a) < d \\ O(n^d log(n)) & log_b(a) = d \\ O(n^{log_b(a)}) & log_b(a) > d \end{cases}$$

Example 1)

Evaluate the runtime of the following recurrence relation:

$$T(n) = 2T(n/4) + n^2$$

Looking at it briefly we see the following.

$$a = 2, b = 4, d = 2$$

$$log_4(2) < 2$$

$$\frac{1}{2} < 2$$

we then assert that by case 1 of master's theorem, we can make a conclusion about T(n)

$$T(n) = O(n^2)$$

note that in this example the recursive portion of the runtime actually doesn't affect it's asymtotic complexity. But in an example where b was a smaller number, it might.

Example 2)

Evaluate the runtime of the following recurrence relation:

$$T(n) = 32T(n/2) + n^5$$

Looking at it briefly we see the following.

$$a = 32, b = 2, d = 5$$

$$log_2(32) = 5$$

$$5 = 5$$

we then assert that by case 2 of master's theorem, we can make a conclusion about T(n)

$$T(n) = O(n^5 log(n))$$