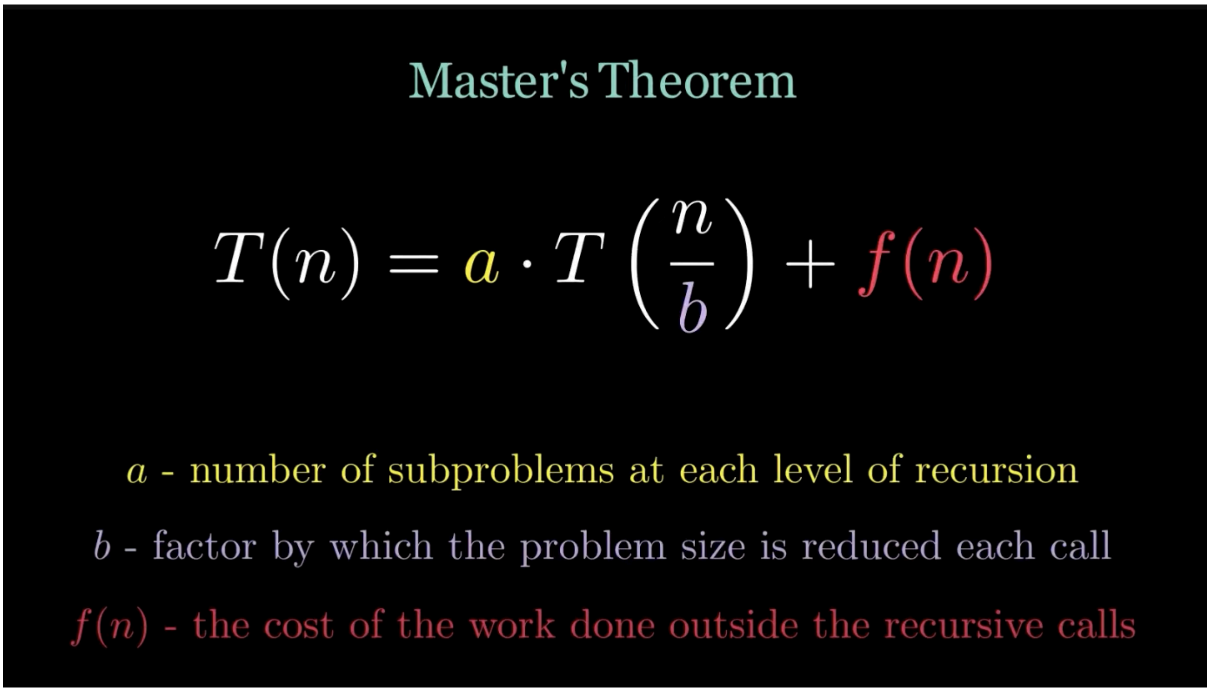
Master’s theorem

It is a method to determine the time complexity of divide-and-conquer algorithms by comparing the work done in recursive calls with the work done outside them.
In the recurrence relation of the algorithms, the non-recursive work done outside the recursive call is represented by **f(n)**.

→ **Master Theorem for Dividing Functions:**



$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$a \geq 1 \quad b > 1 \quad f(n) = O(n^k \log^p n)$$

Here **f(n)** represents the **non-recursive work** done outside the recursive calls.

Here are the two parameters to be considered:

1. $\log_b a$
2. k

Case 1: if $\log_b a > k$ then $O(n^{\log_b a})$

Case 2: if $\log_b a = k$

- if $p > -1$ $O(n^k \log^{p+1} n)$
- if $p = -1$ $O(n^k \log \log n)$
- if $p < -1$ $O(n^k)$

Case 3: if $\log_b a < k$

- if $p \geq 0$ $O(n^k \log^p n)$
- if $p < 0$ $O(n^k)$

Examples:

Case 1:

$$T(n) = 2T\left(\frac{n}{2}\right) + 1$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

Case 2:

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

Case 3:

$$T(n) = T\left(\frac{n}{2}\right) + n^2$$

→ **Master Theorem for Decreasing Functions:**

$$T(n) = aT(n-b) + f(n)$$

$$a > 0, \quad b > 0, \quad f(n) = O(n^k) \quad \text{where } k \geq 0$$

Here **f(n)** represents the **non-recursive work** done outside the recursive calls.

Case 1: If $a < 1$

$$T(n) = O(n^k) \quad \text{or} \quad O(f(n))$$

Case 2: If $a = 1$

$$T(n) = O(n^{k+1}) \quad \text{or} \quad O(n \cdot f(n))$$

Case 3: If $a > 1$

$$T(n) = O\left(n^k \cdot a^{n/b}\right) \quad \text{or} \quad O\left(f(n) \cdot a^{n/b}\right)$$

Examples for Decreasing Functions:

1. $T(n) = T(n-1) + 1$
2. $T(n) = T(n-1) + n$
3. $T(n) = T(n-1) + \log n$
4. $T(n) = 2T(n-2) + 1$
5. $T(n) = 3T(n-1) + 1$
6. $T(n) = 2T(n-1) + n$

Time Complexity of the above Functions:

Time complexity sheet:
https://drive.google.com/file/d/1Tt27gOQLSQn9IfUUQzo-dYkEnCXCsLWT/view?usp=sharing