The Master Theorem

Remington Greko, Tyler Gutowski, and Spencer Hirsch

March 1, 2023

Work with your team to write a report about the *Master Theorem* for solving a recurrence.

I've usually avoided the *Master Theorem* because it has too many rules and does not offer real insight to the solution of the recurrene. However, I do include it in my handouts and the *Master Therorem* may be useful for find the solution to *Strassen's matrix multiplication recurrence.*

$$T(n) = 7T(n/2) + O(n^2)$$

There are many details beneath this recurrence. I strongly encourage you to read what is in the Corman textbook and other sources.

Submit the team's report on Canvas. Include a task matric indicating who did what.

Wikipedia Summary of the Master Theorem

In 1980, the Master Theorem was proposed as the "unifying method" for solving recurrences by Jon Bentley, Dorothea Blostein, and James B. Saxe.[1] Although the name may imply it can solve all recurrences, this is not the case, it's a generalized theory that may be helpful in solving recurrence relations. The Master Theorem utilizes a divide and conquer approach with allows it to separate processing into numerous parts. The master theorem can be expressed by adding the time taken by the top level process with the time made with the recursive calls of the algorithm.[2] The algorithm for the recurrence relation is expressed as:[3]

$$T(n) + aT(n/b) + f(n)$$

Where, n is the input size, a is the number of subproblems, and b is the factor by which the size is reduced.[4] It can be determined based on the Theorem the best case time complexity of the Master Theorem is constant time or *O(1)*.

Brilliant Article on the Master Theorem

Statement of the Master Theorem

Given the algorithm listed above, the runtime of each of the initial nodes is the runtime of T(n/b). Therefore the depth of the tree is $log_b n$ with the depth i has a^i nodes in that final level.[5] Therefore,

$$a^{log_b n} = n^{log_b n}$$

which demonstrates the the runtime of the program is $O(n^{log_b a})$.[6] Through this reasoning you can determine that the form of T is based between f and $n^{log_b a}$.[7]

[1] Wikipedia.
[2] Id.
[3] Id.
[4] Id.
[5] Brilliant
[6] Id.
[7] Id.

The Master Theorem Itself

Given the previous form listed in the Wikipedia section, with constants $a \geq 1$ and $b > 1$ with f.[8] The follwing are true:

"$f(n) = O(n^{log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = O(n^{log_b a})$.

$f(n) = O(n^{log_b a})$, then $T(n) = O(n^{log_b a} logn)$.

$f(n) = O(n^{log_b a + \epsilon})$ for some $\epsilon > 0$, then $T(n) = O(f(n))$." [9]

[8] Id.
[9] Id.

1 Works Cited

"Master Theorem (Analysis of Algorithms)." Wikipedia. Wikimedia Foundation, January 31, 2023. $https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)$.

"Master Theorem." Brilliant Math & Science Wiki. Accessed February 28, 2023. $https://brilliant.org/wiki/master$ $theorem/$.

Name	**Section**
Remington Greko	
Tyler Gutowski	
Spencer Hirsch	Read Wikipedia Article and Brilliant Article