CMSC 351: The Master Theorem

Justin Wyss-Gallifent

March 11, 2024

1 The Theorem (Straightforward Version)

Of course we would rather not do this sort of calculation every time so we might ask if there are reliable formulas which emerge in specific situations and the answer is yes, and these are encapsulated in the Master Theorem:

Theorem 1.0.1. Suppose $T(n)$ satisfies the recurrence relation:

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

for positive integers $a \geq 1$ and $b > 1$ and where $\frac{n}{b}$ can mean either $\left\lfloor \frac{n}{b} \right\rfloor$ or $\left\lceil \frac{n}{b} \right\rceil$, it doesn't matter which. Then we have:

1. If $f(n) = \mathcal{O}(n^c)$ and $\log_b a > c$ then $T(n) = \Theta\left(n^{\log_b a}\right)$.
2. If $f(n) = \Theta(n^c)$ and $\log_b a = c$ then $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$.

2f. (Fancy Version) If $f(n) = \Theta(n^c \lg^k n)$ and $\log_b a = c$ then $T(n) = \Theta\left(n^{\log_b a} \lg^{k+1} n\right)$.

3. If $f(n) = \Omega(n^c)$ and $\log_b a < c$ then $T(n) = \Theta\left(f(n)\right)$.
   Note: For this case, $f(n)$ must also satisfy a *regularity condition* which states that there is some $C < 1$ and n_0 such that $af(n/b) \leq Cf(n)$ for all $n \geq n_0$. This regularity condition is almost always true and we will not worry about it.

Proof. Formal proof omitted. See the intuition section later, if you're interested. $\mathcal{QED}$

2 Application and Examples

When applying the Master Theorem it can be helpful to successively try the cases until we find one that works. It's often easiest to check Case 2 and 2f first as we'll see.

Example 2.1. Consider:

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

Observe that $f(n) = n^2 + \lg n$ and $\log_b a = \log_2 4 = 2$ and then:

- $f(n) = \Theta(n^2)$ with $c = 2$:
  Observe $\log_2 4 = 2 = c$ so Case 2 applies and so:

$$T(n) = \Theta(n^{\log_2 4} \lg n) = \Theta(n^2 \lg n)$$

Example 2.2. Consider:

$$T(n) = 2T(n/8) + n$$

Observe that $f(n) = n$ and $\log_b a = \log_8 2 = 1/3$ and then:

- $f(n) = \Theta(n)$ with $c = 1$:
  Observe $\log_8 2 = 1/3 \neq 1 = c$ so Case 2/2f do not apply.
- $f(n) = \mathcal{O}(n)$ with $c = 1$:
  Observe $\log_8 2 = 1/3 \not> 1 = c$ so Case 1 does not apply.
- $f(n) = \Omega(n)$ with $c = 1$:
  Observe $\log_8 2 = 1/3 < 1 = c$ so Case 3 applies and so:

$$T(n) = \Theta(f(n)) = \Theta(n)$$

Example 2.3. Consider:

$$T(n) = 3T(n/3) + \sqrt{n} + 1$$

Observe that $f(n) = \sqrt{n} + 1$ and $\log_b a = \log_3 3 = 1$ and then:

- $f(n) = \Theta(n^{1/2})$ with $c = 1/2$:
  Observe $\log_3 3 = 1 \neq 1/2 = c$ so Case 2/2f do not apply.
- $f(n) = \mathcal{O}(n^{1/2})$ with $c = 1/2$:
  Observe $\log_3 3 = 1 > 1/2 = c$ so Case 1 applies and so:

$$T(n) = \Theta(n^{\log_3 3}) = \Theta(n)$$

Example 2.4. Consider:

$$T(n) = 3T(n/4) + n \lg n + n$$

Observe that $f(n) = n \lg n + n$ and $\log_b a = \log_4 3 \approx 0.*$ and then:

- $f(n) = \Theta(n \lg n)$ with $c = 1$:
  Observe $\log_4 3 \approx 0.* \neq 1 = c$ so Case 2/2f do not apply.
- $f(n) = \mathcal{O}(n^2)$ with $c = 2$:
  Observe $\log_4 3 \approx 0.* \not> 2 = c$ so Case 1 does not apply.
- $f(n) = \Omega(n)$ with $c = 1$:
  Observe $\log_4 3 \approx 0.* < 1 = c$ so Case 3 applies and so:

$$T(n) = \Theta(f(n)) = \Theta(n \lg n)$$

Example 2.5. Consider:

$$T(n) = 10T(n/2) + n^2 \lg n + n^2 + 1$$

Observe that $f(n) = n^2 \lg n + n^2 + 1$ and $\log_b a = \log_2 10 \approx 3.*$ and then:

- $f(n) = \Theta(n^2 \lg n)$ with $c = 2$:
  Observe $\log_2 10 \approx 3.* \neq 2 = c$ so Case 2/2f do not apply.
- $f(n) = \mathcal{O}(n^3)$ with $c = 3$:
  Observe $\log_2 10 \approx 3.* > 3 = c$ so Case 1 applies and so:

$$T(n) = \Theta(n^{\log_2 10})$$

Example 2.6. Consider:

$$T(n) = 9T(n/3) + n^2 \lg n + \lg n$$

Observe that $f(n) = n^2 \lg n + \lg n$ and $\log_b a = \log_3 9 = 2$ and then:

- $f(n) = \Theta(n^2 \lg n)$ with $c = 2$:
  Observe $\log_3 9 = 2 = c$ so Case 2f applies with $k = 1$ and so:

$$T(n) = \Theta(n^{\log_3 9} \lg^2 n) = \Theta(n^2 \lg^2 n)$$

Here is a slightly tricker example:

Example 2.7. Consider:

$$T(n) = 16T(n/2) + n^3 \lg n$$

Observe that $f(n) = n^3 \lg n$ and $\log_b a = \log_2 16 = 4$. Now observe:

- $f(n) = \Theta(n^3 \lg n)$ with $c = 2$ and $k = 1$ so it looks like Case 2f with $c = 3$ and $k = 1$ but $\log_b a = 4 \neq 3 = c$ so it's not Case 2f.
- $f(n) = \mathcal{O}(n^4)$ with $c = 4$ but $\log_b a = 4 \not> 4 = c$ so it's not Case 1. Or it is?
- $f(n) = \mathcal{O}(n^{3.5})$ (as can be proven with the limit theorem) with $c = 3.5$ and $\log_b a = 4 > 3.5$ so it is in fact Case 1 when treated carefully and so:

$$T(n) = \Theta(n^{\log_2 16}) = \Theta(n^4)$$

Here are some examples where it does not apply:

Example 2.8. Supppose $T(n) = 2T(n/4) + 3T(n/2) + n$.

The Master Theorem does not apply because it has the wrong form. Note that there is another method which often applies called the Akra-Bazzi method. It applies to recurrence formulas of the following form under certain conditions:

$$T(n) = f(n) + \sum_{i=1}^{k} a_i T(b_i n + h_i(n))$$

We will not cover it.

Example 2.9. Supppose $T(n) = 2T(n/4) + f(n)$ and all you know is that $f(n) = \mathcal{O}(n^2)$.

The fact that $f(n) = \mathcal{O}(n^2)$ implies that we could only use Case 1 and insists that $c = 2$. However $\log_b a = \log_4 2 = 0.5 \not> 2 = c$ and so Case 1 does not apply.

Example 2.10. Supppose $T(n) = 8T(n/4) + f(n)$ and all you know is that $f(n) = \Omega(n)$.

The fact that $f(n) = \Omega(n)$ implies that we could only use Case 3 and insists that $c = 1$. However $\log_b a = \log_4 8 = \frac{3}{2} \not< 1 = c$ and so Case 3 does not apply.

3 Motivation Behind the Theorem

3.1 Intution Without the $f(n)$

If $f(n) = 0$ then $f(n) = \mathcal{O}(1) = \mathcal{O}(n^0)$ and $0 < \log_b a$ as long as $a > 1$ (which it is) and so all what follows here lies in the first case of the Master Theorem.

Consider a divide-and-conquer algorithm which breaks a problem of size n into a subproblems each of size n/b. In such a case we would have:

$$T(n) = aT(n/b)$$

Now observe:

- It seems reasonable that if $a = b$ then we have no overall gain because the number of new problems equals the reducing ratio (for example two problems of half the size doesn't help) but we can actually say more.

  If we assume a reasonable $T(1) = \alpha$ for some constant α then this is essentially saying, for example, that $T(2) = 2T(2/2) = 2(1) = 2\alpha$, $T(4) = 2T(4/2) = 2(2) = 4\alpha$, $T(8) = 2T(8/2) = 2(4) = 8\alpha$, and so on, and in general it seems reasonable that $T(n) = n\alpha = \Theta(n)$.

  This also seems reasonable with any $a = b$ (not just 2), that we still get $T(n) = \Theta(n)$.

  This arises in the first case of the Master Theorem because if $a = b$ then $\log_b a = 1$ and then $T(n) = \Theta(n^{\log_b a}) = \Theta(n^1)$.

- On the other hand if $b > a$ then we have an overall decrease in time, for example if $T(n) = 2T(n/3)$ then the subproblems are $1/3$ the size and there are only two, that's good, better than $\Theta(n)$!

  This arises in the first case of the Master Theorem because if $b > a$ then $\log_b a < 1$ and then $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\text{less than } 1})$.

- And on the other hand if $b < a$ then we have an overall gain in time,, for example if $T(n) = 3T(n/2)$ then the subproblems are $1/2$ the size but there are three, that's bad, worse than $\Theta(n)$!

  This arises in the first case of the Master Theorem because if $b < a$ then $\log_b a > 1$ and then $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\text{more than } 1})$.

3.2 Proof Without the $f(n)$

If $f(n) = 0$ we can in fact solve the recurrence relation easily using the digging-down approach. If we accept a base case of $T(1)$ then we have:

$$\begin{aligned} T(n) &= aT(n/b) \\ &= a^2 T(n/b^2) \\ &= a^3 T(n/b^2) \\ \vdots \;\; & \;\; \vdots \end{aligned}$$

For any $k \geq 1$ we have $T(n) = a^k T(n/b^k)$ which ends when $n/b^k = 1$ which is $k = \log_b n$.

Thus we get:

$$\begin{aligned} T(n) &= a^k T(1) \\ &= a^{\log_b n} T(1) \\ &= a^{(\log_a n / \log_a b)} T(1) \\ &= \left(a^{\log_a n}\right)^{1/\log_a b} T(1) \\ &= n^{1/\log_a b} T(1) \\ &= n^{\log_b a} T(1) \\ &= \Theta\left(n^{\log_b a}\right) \end{aligned}$$

3.3 Intution with the $f(n)$

Now that we've intuitively and formally accepted that:

$$T(n) = aT(n/b) \quad \Longrightarrow \quad T(n) = \Theta(n^{\log_b a})$$

Now let's suppose there is some additional time requirement $f(n)$ for a problem of size n.

- If this new time requirement is at most (meaning $\mathcal{O}$) polynomially smaller than the recursive part then the recursive part is the dominating factor. This is represented in the theorem by the line:

  If $f(n) = \Theta(n^c)$ for $c < \log_b a$ then $T(n) = \Theta\left(n^{\log_b a}\right)$.

- If this new time requirement is the same (meaning Θ) polynomially as the recursive part then they combine and a logarithmic factor is introduced (this is not obvious). This is represented in the theorem by the line:

  If $f(n) = \Theta(n^c)$ for $c = \log_b a$ then $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$.

- If this new time requirement is at least (meaning Ω)mpolynomially larger than the recursive part then this new time requirement is the dominating factor. This is represented in the theorem by the line:

  If $f(n) = \Theta(n^c)$ for $c > \log_b a$ then $T(n) = \Theta\left(f(n)\right)$.

  Along with the regularity condition.

4 Even More Rudimentary Intuition

To think even more crudely, but not inaccurately, when you see the recurrence relation:

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

Think of the $aT(n/b)$ as the tree structure and $f(n)$ as the node weights. Basically the Master Theorem is saying:

1. If the tree structure is greater than the node weights then the node weights don't matter and the tree structure wins - winter.
2. If they balance out nicely then they combine - spring.
3. If the tree structure is less than the node weights then the tree structure doesn't matter and the node weights win - summer.

5 Thoughts, Problems, Ideas

1. Suppose $T(n) = 5T(n/5) + f(n)$.
   (a) Apply the Master Theorem with $f(n) = \sqrt{n}$.
   (b) Apply the Master Theorem with $f(n) = n + \sqrt{n}$.
   (c) Apply the Master Theorem with $f(n) = 3n + \sqrt{n^3}$.
   (d) Apply the Master Theorem with $f(n) = n \lg n$.
2. Suppose $T(n) = 4T(n/8) + f(n)$.
   (a) Apply the Master Theorem with $f(n) = \sqrt{n}$.
   (b) Apply the Master Theorem with $f(n) = n^{2/3} + \lg n$.
   (c) Apply the Master Theorem with $f(n) = n$.

3. Suppose $T(n) = 4T(n/2) + f(n)$.
   (a) Apply the Master Theorem with $f(n) = n + \sqrt{n}$.
   (b) Apply the Master Theorem with $f(n) = n^2 + n + 1$.
   (c) Apply the Master Theorem with $f(n) = n^3 \lg n$.
4. Binary Search has $T(n) = T(n/2) + \Theta(1)$. What is $T(n)$?
5. Merge Sort has $T(n) = 2T(n/2) + \Theta(n)$. What is $T(n)$?
6. The Max-Heapify routine in Heap Sort has $T(n) \leq T(2n/3) + \Theta(1)$. What is $T(n)$?
7. An optimal sorted matrix search has $T(n) = 2T(n/2) + \Theta(n)$. What is $T(n)$?
8. A divide and conquer algorithm which splits a list of length n into two equally sized lists, makes recursive calls to both and in addition uses constant time will have $T(n) = 2T(n) + C$. What is $T(n)$?