Master Theorem (M.T.)

If we can express the running time of an algorithm by the recurrence

\[ T(n) = a T\left(\frac{n}{b}\right) + f(n) \]

where \( a \geq 1 \), \( b > 1 \) are constants, and we interpret \( \frac{n}{b} \) as either \( \left\lfloor \frac{n}{b} \right\rfloor \) or \( \left\lceil \frac{n}{b} \right\rceil \), then we can bound \( T(n) \) under certain conditions as follows:

1. \( f(n) = O(n^{\log_a b - \varepsilon}) \) for some const \( \varepsilon > 0 \)
   \[ \Rightarrow T(n) = \Theta(n^{\log_a b}) \]

2. \( f(n) = \Theta(n^{\log_a b}) \)
   \[ \Rightarrow T(n) = \Theta(n^{\log_a b} \log n) \]

3. \( f(n) = \Omega(n^{\log_a b} + \varepsilon) \) for some const \( \varepsilon > 0 \)
   and \( af\left(\frac{n}{b}\right) \leq cf(n) \) for some const \( c < 1 \) and sufficiently large \( n \)
   \[ \Rightarrow T(n) = \Theta(f(n)) \]

Case 2) is "middle ground". When trying to apply Master theorem, it is useful to consider case 2) first, i.e. compare \( f(n) \) with \( n^{\log_a b} \). If \( f(n) = \Theta(n^{\log_a b}) \) we are at right place, otherwise we can consider case 3) or case 1) depending on whether \( f(n) \) is asymptotically "bigger" or "smaller" than \( n^{\log_a b} \), respectively.

Example 1. Merge Sort

\[ \begin{array}{c}
\text{split} \quad c_1 = \Theta(1) \\
\downarrow \\
\text{sort } T(n/2) \quad \text{sort recursively } 2T\left(\frac{n}{2}\right) \\
\downarrow \\
\text{merg}e \quad c_2 = \Theta(n)
\end{array} \]
\[ T(n) = 2T\left(\frac{n}{2}\right) + f(n) \quad (\text{where} \quad f(n) = C_1 + C_2 = \Theta(1)) \]

**Step 1:** What are \( a \) and \( b \)?

\( a = 2, \quad b = 2 \)

**Step 2:** Compute \( n^{\log_b a} \)

\[ n^{\log_b a} = n^{\log_2 2} = n^1 = n \]

**Step 3:** Compute \( f(n) \) with \( n^{\log_b a} \)

\[ f(n) = \Theta(n) = \Theta(n^{\log_b a}) \]

\[ \Rightarrow \text{we can apply case 2) of M.T.} \]

\[ \Rightarrow T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n \log n) \]

**Example 2: Binary Search**

- **Base Case:** \( T(\frac{n}{2}) \)
- **Recursive Case:** \( T(n) = 2T\left(\frac{n}{2}\right) + f(n) \)

**Step 1:** \( a = 1, \quad b = 2 \)

**Step 2:** \( n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \)

**Step 3:** \( f(n) = \Theta(1) = \Theta(n^{\log_b a}) \)

\[ \Rightarrow (\text{by M.T. 2}) \quad T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n \log n) \]

**Note:** This analysis apply if we recapture until problem is reduced to a single element, i.e., in the worst case, so we can write \( T(n) = \Theta(\log n) \) in the worst case. However, we may terminate search earlier, and in the best case we will terminate after first step. Therefore, we can write \( T(n) = O(\log n), \quad T(n) = \Omega(1) \) for any case.
Example 3. 1-d Peak Finder

Similar to binary search.

\[ T(n) = T\left(\frac{n}{2}\right) + f(n) \text{, where } f(n) = c_1 + c_2 = \Theta(1) \]

\[ \Rightarrow T(n) = \Theta(\log n) \]

Example 4. 2-d Peak Finder

\[ T(n) = T\left(\frac{n}{2}\right) + f(n) \text{, where } f(n) = 6c_1n + c_2 + c_3 = \Theta(n) \]

\[ \text{step 1: } a = 1, b = 2 \]
\[ \text{step 2: } n^{\log a} = n^{\log 1} = n^0 = 1 \]
\[ \text{step 3: } f(n) = \Theta(n) \neq \Theta(n^{\log a}) \Rightarrow \text{we cannot apply M.T.2} \]

However, \( f(n) = \Omega(n^{\log a}) \), so we can try to apply M.T.3.

\[ \text{step 4: choose } c > 0 \text{ such that } f(n) = \Omega(n^{\log a + \epsilon}) \]

\[ \text{Apprently, this will hold for any } \epsilon \leq 1 \text{, so let's choose } \epsilon = 1 \Rightarrow f(n) = \Omega(n^{\log a + 1}) \]

\[ \text{step 5: find } c < 1 \text{, such that } af\left(\frac{n}{b}\right) \leq cf(n) \text{ for sufficiently large } n \]

\[ f(n) = c_1n + c_2 \]
\[ c_1\frac{n}{b} = f\left(\frac{n}{b}\right) = c_1\frac{n}{b} + c_2 \]
\[ cf(n) = c \cdot (c_1n + c_2) \]
Apparently, \( af(b) > cf(n) \) for any \( c \leq \frac{a}{b} \).

However, for \( c > \frac{a}{b} \) coefficient that multiplies \( n \) is bigger. If \( cf(n) \) then in \( af(b) \), so for sufficiently large \( n \), \( af(b) \leq cf(n) \) will hold.

E.g., choose \( c = \frac{3}{4} \):

\[
\Rightarrow \quad cf(n) = \frac{3}{4} c_1 n + \frac{3}{4} c_2
\]

Find \( n \) s.t. \( af(b) \leq cf(n) \):

\[
\Rightarrow \quad c_1 \frac{a}{b} n + c_2 \leq \frac{3}{4} c_1 n + \frac{3}{4} c_2
\]

\[
\Rightarrow \quad c_1 \frac{a}{b} n + c_2 \leq \frac{3}{2} c_1 \frac{a}{b} \frac{n}{2} + \frac{3}{4} c_2
\]

\[
\Rightarrow \quad \frac{1}{4} c_2 \leq \frac{1}{4} c_1 \frac{a}{b} \frac{n}{2}
\]

\[
\Rightarrow \quad n \geq \frac{c_2}{c_1 \frac{a}{b}}
\]

\[
\Rightarrow \quad \text{for } c = \frac{3}{4}, \text{and sufficiently large } n, \quad af(b) \leq cf(n)
\]

Finally, we can apply A.T. 3:

\[
\Rightarrow \quad T(n) = \Theta(f(n)) = \Theta(n)
\]

---

**Example 5. Compute Hash Values on Binary (Complete) Tree (from ASET 2)**

1. **Sequence of Lines**
   - Split into two halves: \( c_1 = \Theta(1) \)
   - Compute hash values at each half: \( 2T(\frac{n}{2}) \)
   - Combine two hash values: \( c_2 = \Theta(1) \)
   - (Note: By combining hash values of subnodes, we do get \( \Theta(1) \) for computing each node's hash value; earlier to finding has

\[
T(n) = 2T(\frac{n}{2}) + f(n), \text{ where } f(n) = c_1 + c_2 = \Theta(1)
\]

- **Step 1:** \( a = 2, b = 2 \)
- **Step 2:** \( n^{\log a} = n^{\log 2} = n = n \)
- **Step 3:** \( f(n) = \Theta(1) \neq \Theta(n^{\log a}) \)

However, \( f(n) = O(n^{\log a}) \), so we can try to apply A.T. 1.

- **Step 4:** Choose \( \varepsilon > 0 \), s.t. \( f(n) = O(n^{\log a - \varepsilon}) \)

We can choose any \( \Theta \leq \varepsilon \leq 1 \), let's choose \( \varepsilon = 1 \)

\[
f(n) = \Theta(n) = \Theta(n^0) = \Theta(n^{\log a - \varepsilon}) \implies f(n) = O(n^{\log a - \varepsilon})
\]

\[
\Rightarrow \quad (by \text{ A.T.1}) \quad T(n) = \Theta(n^{\log a}) = \Theta(n)
\]
Example 6. \( T(n) = 2T\left(\frac{n}{2}\right) + n^2 \)

\[ \begin{align*}
    a &= 2 \\
    b &= 2 \\
    \log_b a &= \log_2 2 = 1
\end{align*} \]

\( h = n \)

\( \delta &= \log_b a + 1 \)

\( \Rightarrow \) for any \(\delta \), \( n = \Omega\left(n^{\log_b a + \delta}\right) \)

\[ \Rightarrow \quad a f\left(\frac{n}{2}\right) = 2 \cdot \left(\frac{n}{2}\right)^2 = \frac{n^2}{2} \quad \Rightarrow \quad a f\left(\frac{n}{2}\right) \leq c f(n) \quad \text{for} \quad c = \frac{1}{2} \quad (\text{for any} \quad n) \]

\( \text{e.g. take} \quad c = \frac{1}{2} \)

\( \Rightarrow \quad \text{M.T.3)} \quad T(n) = \Theta(f(n)) = \Theta(n^2) \)

Example 7. \( T(n) = 9T\left(\frac{n}{3}\right) + n \)

\[ \begin{align*}
    a &= 9 \\
    b &= 3 \\
    \log_b a &= \log_3 9 = 2
\end{align*} \]

\( f(n) = n = O(n) = O\left(n^{\log_b a - \varepsilon}\right) \quad \text{e.g. for} \quad \varepsilon = 0 \quad (\text{again, any} \quad 0 < \varepsilon \leq 1 \quad \text{will work}) \)

\( \Rightarrow \quad \text{by M.T.1)} \quad T(n) = \Theta\left(n^{\log_b a}\right) = \Theta(n^2) \)

Example 8. \( T(n) = 3T\left(\frac{n}{4}\right) + n \log n \)

\[ \begin{align*}
    a &= 3 \\
    b &= 4 \\
    \log_b a &= \log_4 3 \approx 0.793
\end{align*} \]

\( \omega(n \log n) = \Omega(n) = \Omega\left(n^{\log_4 3 + \varepsilon}\right) \), for any \( 0 < \varepsilon \leq \frac{1}{2} \); e.g. take \( \varepsilon = 0.12 \)

\[ \begin{align*}
    a f\left(\frac{n}{4}\right) &= 3 \cdot \left(\frac{n}{4}\right) \cdot \log \left(\frac{n}{4}\right) = \frac{3}{4} n \log n - \frac{3}{4} n \log 4 = \frac{3}{4} n \log n - \frac{3}{4} n \cdot 2 = \frac{3}{4} n \log n - \frac{3}{2} n
\end{align*} \]

\( cf(n) = cn \log n \)

Is there \( c_1 \) such that \( a f\left(\frac{n}{4}\right) = \frac{3}{4} n \log n - \frac{3}{2} n \leq c n \log n = cf(n) \) ? (for sufficient \( n \))

Obviously, inequality holds for any \( n \) if \( c \geq \frac{3}{4} \), so let's take \( c = \frac{3}{4} \).

\( \Rightarrow \quad \text{M.T.3)} \quad T(n) = \Theta(f(n)) = \Theta(n \log n) \)
Example 9. \( T(n) = 2T\left(\frac{n}{2}\right) + n \log n \)

\[ a = 2, \quad b = 2 \]

\[ \log_2 a = \log_2 2 = 1 \quad \Rightarrow \quad n^{\log_2 a} = n \]

\( f(n) = n \log n \)

Obviously \( f(n) \neq \Theta(n) \)

It is rather: \( f(n) = \Omega(n) \), so we cannot apply case 1) or case 2) of N.T. 1 but \( \Rightarrow \) it seems that we can apply case 3).

However, even though \( f(n) = \Omega(n) \), there is no \( \varepsilon > 0 \) such that \( f(n) = \Omega(n^{1+\varepsilon}) \).

That is because \( f(n) \) is not polynomially larger than \( n^{\log_2 a} = n \).

\[ \frac{f(n)}{n} = \frac{n \log n}{n} = \log n \quad \text{which is smaller than} \quad n^{\varepsilon} \quad \text{for any} \quad \varepsilon > 0 \quad \text{asymptotically}, \]

\[ \lim_{n \to \infty} \frac{\log n}{n^\varepsilon} = \lim_{n \to \infty} \frac{\varepsilon \log n}{\varepsilon n^\varepsilon} = \lim_{n \to \infty} \frac{\log n}{n^{1+\varepsilon}} = \lim_{x \to \infty} \frac{\log x}{x^{1+\varepsilon}} = 0 \]

Therefore, we cannot apply N.T. 3).

Alternatively, we could show that there is no \( c < 1 \) s.t. \( af(n) < cf(n) \) for large \( n \).

\[ af\left(\frac{n}{2}\right) = \frac{2f(n)}{2} = n \log \frac{n}{2} = n \log n - n \log 2 = n \log n - n \]

\[ cf(n) = C n \log n \]

\[ af(n) \leq cf(n) \]

\[ \Rightarrow \quad n \log n - n \log 2 \leq C n \log n \]

\[ \Rightarrow \quad n \log (1 - \frac{1}{c}) \leq n \log n \]

\[ \Rightarrow \quad (1 - \frac{1}{c}) \leq 1 \]

However, for \( c < 1 \), \( 1 - c > 0 \) for large enough \( n \) (\( n \geq 2 \))

Inequality does not hold.

So, we cannot apply N.T. 3)

What can we do then?

We can try some other method than N.T. (for example, we can try to solve it the same way we did before knowing N.T.) or we can apply some extensions of N.T. that have been developed.
Solution by "unrolling" $T(n)$ (as we did before)

\[ T(n) = 2T(\frac{n}{2}) + n\log n \quad \text{I assume that } n = 2^k, k = \log n \]
\[ = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\log \frac{n}{2}\right) + n\log n \]
\[ = 4T\left(\frac{n}{4}\right) + n\log \frac{n}{2} + n\log n \]
\[ = 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\log \frac{n}{4}\right) + n\log \frac{n}{2} + n\log n \]
\[ = 8T\left(\frac{n}{8}\right) + n\log \frac{n}{8} + n\log \frac{n}{2} + n\log n \]
\[ = \ldots = nT(n) + n\left(\log 2 + \log 4 + \ldots + \log \frac{n}{2} + \log n\right) \]
\[ = nT(n) + n\left(\log 2^2 + \log 2^4 + \ldots + \log 2^{k-1} + \log 2^k\right) \]
\[ = nT(n) + n\left(1 + 2 + \ldots + k-1 + k\right) \]
\[ = nT(n) + n\frac{k(k+1)}{2} \]
\[ = nT(n) + n\cdot \frac{1}{2} \log n \left(\log n + 1\right) \]

Since $T(1) = \Theta(1)$,

\[ T(n) = n\Theta(n) + \Theta(\log^2 n) \]
\[ = \Theta(n\log^2 n) \]

---

**Extended Master Theorem (E.M.T.)**

*You are not required to know E.M.T. in this course!*

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

1) If $f(n) = O(n^\log_b a) \log^d n$ with $\alpha < -1$

   \[ \Rightarrow T(n) = \Theta(n^\log_b a) \]

2) If $f(n) = \Theta\left(n^\log_b a \log^\alpha n\log n\right)$

   \[ \Rightarrow T(n) = \Theta\left(n^\log_b a \log \log \log n\right) \]

3) If $f(n) = \Theta\left(n^\log_b a \log^\alpha n\log n\right)$ with $\alpha \geq -1$

   \[ \Rightarrow T(n) = \Theta(n^\log_b a \log^\alpha n\log n) \]

4) Same as M.T.3, i.e., if $f(n) = \Omega(n^\log_b a \log^\alpha n)$ for some $\alpha$ and there is $N \leq f(n)$ by (suff the $N(n)$)
Solution to example 9. by applying E.M.T.

\[ T(n) = 2T\left(\frac{n}{2}\right) + \lg n \]
\[ a = 2, b = 2 \]
\[ n^{\log_a^b} = n^{\frac{\log_2 2}{1}} = n^1 = n \]

\[ f(n) = n \lg n = \Theta(n \lg n) = \Theta(n^{\log_a^b (\log \log n)}^{\frac{1}{1}}) = \Theta(n^{\log_a^b (\log \log n)}) \text{ for } b = 1 \]
\[ \Rightarrow \text{ we can apply E.M.T. 3) } \]
\[ \Rightarrow T(n) = \Theta(n^{\log_a^b (\log \log n)^{a+b}}) = \Theta(n^{\log_2^2 (\log \log n)^{2+1}}) = \Theta(n \log^2 n) \]

Note that this example \( f(n) \) falls somewhere in between cases 2) and 3) of M.T. 1 and therefore E.M.T. 2) provides solutions to the recurrence for some more families of functions \( f(n) \) not considered in M.T.

Example 10. Different log basis: constants matter if everything else is the same

Asymptotic time complexity of merge sort is \( \Theta(n \log n) \).

Let's try the following modification: Instead of 2, split array into 3 equal size subarray, sort each recursively and then merge them using a 3-fingers method (always advance the fingers that points to the smallest element of the 3 elements).

Let's call it temporary merge sort.

\[ T(n) = 3T\left(\frac{n}{3}\right) + f(n) \text{ where } f(n) = C_3^2 \cdot n + C_3^1 \]
\[ a = 3, b = 3 \]
\[ n^{\log_a^b} = n^{\frac{\log_3 3}{1}} = n^1 = n \]
\[ \Rightarrow f(n) = \Theta(n) = n^{\log_a^b} \]
\[ \Rightarrow \text{ M.T. 2) } T(n) = \Theta(n \log n) \]
While both binary and ternary merge sort have the same asymptotic time complexity, the constants that we omit from "Oh" notation might be different, and therefore, one might be favorable.

Since Master theorem does not involve constants, we can go back to the method we used before, and write $T(n)$ with constants included.

**Binary Merge Sort**

$$T^2(n) = 2T^2(n/2) + C_1^2 n + C_1$$

where $C_1$ is cost of splitting, and $C_2^2 n$ cost of merging.

$$T^2(n) = 2(2T^2(n/4) + C_2^2 n/2 + C_1^2) + C_2^2 n + C_1^2$$

$$= 4T(n/4) + C_2^2 n + C_2^2 n/2 + 2C_1^2 + C_1^2$$

$$= \cdots = nT(n) + \underbrace{C_2^2 n + \cdots + C_2^2 n}_{\log_2 n \text{ times}}$$

$$= nT^2(n) + C_2^2 n \log_2 n + (n-1) C_1^2$$

**Ternary Merge Sort**

$$T^3(n) = 3T^3(n/3) + C_3^3 n + C_1^3$$

where $C_3$ is cost of splitting, and $C_3^3 n$ is cost of merging.

$$T^3(n) = 3(3T^3(n/9) + C_3^3 n/3 + C_1^3) + C_3^3 n + C_1^3$$

$$= 9T^3(n/9) + C_3^3 n + C_3^3 n/3 + 3C_1^3 + C_1^3$$

$$= \cdots = nT(n) + \underbrace{C_3^3 n + \cdots + C_3^3 n}_{\log_3 n \text{ times}}$$

$$= nT^3(n) + C_3^3 n \log_2 n + \frac{n-1}{2} C_3^3$$

$$= nT^3(n) + C_3^3 n \log_2 n + \frac{n-1}{2} C_3^3$$

In both expressions (binary and ternary), $n \log_2 n$ is the most significant factor—asymptotically larger than other factors, so for large $n$ it is the only one that matters.

$$T^2(n) \propto C_2^2 n \log_2 n$$

$$T^3(n) \propto C_3^3 \log_3 n$$

Thus, we need to compare $C_2^2$ and $C_3^3 \log_3 n$.

Since merging in ternary merge sort requires finding min of 3 elements and merging in binary merge sort requires finding min of 3 elements, it is $C_3^3 > C_2^2$, but we don't know exact nature because there are other operations involved and we need to know cost model of the programming language.