Master Theorem (M.T.)

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

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

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

M.T.1)) 1) $f(n) = O(n^{\log_b a - \varepsilon})$ for some const $\varepsilon > 0$

$$\Rightarrow T(n) = \Theta(n^{\log_b a})$$

M.T.2)) 2) $f(n) = \Theta(n^{\log_b a})$

$$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$$

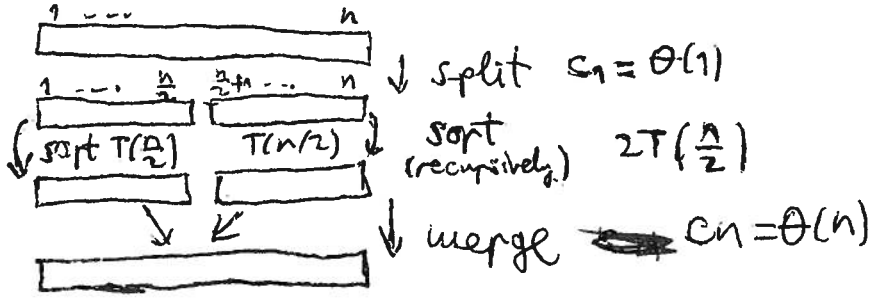
M.T.3)) 3) $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some const $\varepsilon > 0$

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 $n^{\log_b a}$. If $f(n) = \Theta(n^{\log_b a})$ 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 $n^{\log_b a}$, respectively.

Example 1. Merge Sort



$T(n) = \underset{a}{2}T(\frac{n}{\underset{b}{2}}) + f(n)$, where $f(n) = c_1 n + c_1 = \Theta(n)$

Step 1: What are a and b ? (in formula $T(n) = aT(\frac{n}{b}) + f(n)$)

$a = 2$, $b = 2$

step 2: compute $n^{\log_b a}$

$n^{\log_b a} = n^{\log_2 2} = n^1 = n$

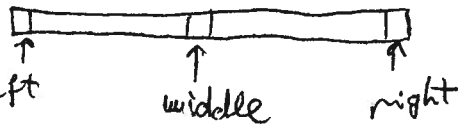
step 3: Compare $f(n)$ with $n^{\log_b a}$

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

$\Rightarrow$ we can apply case 2) of M.T.

$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg n) = \underline{\Theta(n \lg n)}$

Example 2. Binary Search

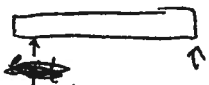


compute middle element - $c_1 = \Theta(1)$

compare middle element to the searched value - $c_2 = \Theta(1)$

either

or



$\Downarrow$ recurse on one half (search) - $T(\frac{n}{2})$

$T(n) = T(\frac{n}{2}) + f(n)$, where $f(n) = c_1 + c_2 = c = \Theta(1)$

step 1: $a = 1$, $b = 2$

step 2: $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$

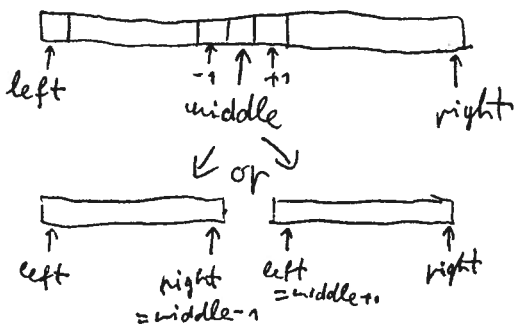
step 3: $f(n) = \Theta(1) = \Theta(n^{\log_b a})$

$\Rightarrow$ (By M.T.2) $T(n) = \Theta(n^{\log_b a} \lg n) = \underline{\Theta(\lg n)}$

Note: This analysis apply if we recurse until problem is reduced to a single element, i.e., in the worst case. So, we can write $T(n) = \Theta(\lg n)$ 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 $T(n) = O(\lg n)$, $T(n) = \Omega(1)$ for any case.

Example 3. 1-d peak Finder

Similar to binary search.



find the middle - $c_1 = \Theta(1)$

compare 3 middle elements - $c_2 = \Theta(1)$

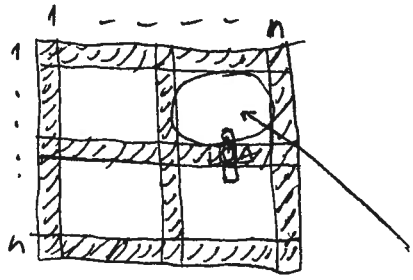
recurse on one half - $T(\frac{n}{2})$

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

↖ same as binary search

$\Rightarrow T(n) = \underline{\Theta(\lg n)}$

Example 4. 2-d peak Finder



find max on 3 rows and 3 columns - $c_1 \cdot 6n + c_2 = \Theta(n)$

choose a quadrant - $c_3 = \Theta(1)$
based on neighbors

recurse on one quadrant - $T(\frac{n}{2})$
of size $\frac{n}{2} \times \frac{n}{2}$

$T(n) = T(\frac{n}{2}) + f(n)$, where $f(n) = 6c_1 n + c_2 + c_3 = \Theta(n)$

step 1: $a = 1$, $b = 2$

step 2: $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$

step 3: $f(n) = \Theta(n) \neq \Theta(n^{\log_b a})$ $\Rightarrow$ we cannot apply M.T.2)

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

step 4: Choose $\varepsilon > 0$, such that $f(n) = \Omega(n^{\log_b a + \varepsilon})$

Apparently, this will hold for any $0 < \varepsilon \leq 1$, so let's choose $\varepsilon = 1 \Rightarrow f(n) = \Omega(n) = \Omega(n^{\log_b a + \varepsilon})$

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

$f(n) = c_1 n + c_2$

$af(\frac{n}{b}) = f(\frac{n}{2}) = c_1 \frac{n}{2} + c_2$

$cf(n) = c \cdot (c_1 n + c_2)$

Apparently, $af(\frac{n}{6}) > cf(n)$ for any $c \leq \frac{1}{2}$

However, for $c > \frac{1}{2}$ coefficient that multiplies n is bigger in $cf(n)$ than in $af(\frac{n}{6})$

so for sufficiently large n, $af(\frac{n}{6}) \leq cf(n)$ will hold.

E.g., choose $c = \frac{3}{4}$

$\Rightarrow cf(n) = \frac{3}{4}c_1 n + \frac{3}{4}c_2$

find n s.t. $af(\frac{n}{6}) \leq cf(n)$

$\Leftrightarrow c_1\frac{n}{2} + c_2 \leq \frac{3}{4}c_1 n + \frac{3}{4}c_2$

$\Leftrightarrow c_1\frac{n}{2} + c_2 \leq \frac{3}{2}\cdot c_1 \cdot \frac{n}{2} + \frac{3}{4}c_2$

$\Leftrightarrow \frac{1}{4}c_2 \leq \frac{1}{2}c_1 \cdot \frac{n}{2}$

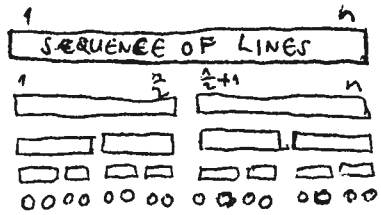
$\Leftrightarrow n \geq \frac{c_2}{c_1}$

$\Rightarrow$ for $c = \frac{3}{4}$ and sufficiently large n, $af(\frac{n}{6}) \leq cf(n)$

Finally, we can apply M.T.3)

$\Rightarrow T(n) = \theta(f(n)) = \underline{\theta(n)}$

Example 5. Compute Hash values on Binary (complete) Tree (from PSET 2)



split into two halves - $c_1 = \theta(1)$

compute hash values of each half - $2T(\frac{n}{2})$

combine ~~two~~ hash values - $c_2 = \theta(1)$

(Note: By ~~combining~~ hash values of subsequences, we ~~must~~ get $\theta(1)$ for computing each node hash value, similar to rolling has

$T(n) = 2T(\frac{n}{2}) + f(n)$, where $f(n) = c_1 + c_2 = \theta(1)$

step 1: $a = 2$, $b = 2$

step 2: $n^{\log_b a} = n^{\log_2 2} = n^1 = n$

step 3: $f(n) = \theta(1) \neq \theta(n^{\log_b a})$

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

step 4: choose $\varepsilon > 0$, s.t. $f(n) = O(n^{\log_b a - \varepsilon})$

we can choose any $0 < \varepsilon \leq 1$, let's choose $\varepsilon = 1$

$f(n) = \theta(1) = \theta(n^0) = \theta(n^{\log_b a - \varepsilon}) \Rightarrow f(n) = O(n^{\log_b a - \varepsilon})$

$\Rightarrow$ (By M.T.1)) $T(n) = \theta(n^{\log_b a}) = \underline{\theta(n)}$

Example 6. $T(n) = 2T(\frac{n}{2}) + n^2$

$a = 2$

$b = 2$

$\log_b a = \log_2 2 = 1$

$f(n) = n^2 = n^{\log_b a + 1}$

$\Rightarrow$ for $\varepsilon = 1$, $n^2 = \Omega(n^{\log_b a + 1})$ (we could choose any $0 < \varepsilon \leq 1$)

$af(\frac{n}{b}) = 2 \cdot (\frac{n}{2})^2 = \frac{n^2}{2}$

$cf(n) = cn^2$

$\Rightarrow af(\frac{n}{b}) \leq cf(n)$ for $c \geq \frac{1}{2}$ (for any n)

e.g. take $c = \frac{1}{2}$

$\Rightarrow$ by M.T.3) $T(n) = \Theta(f(n)) = \underline{\Theta(n^2)}$

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

$a = 9$, $b = 3$

$\log_b a = \log_3 9 = 2$

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

$\Rightarrow$ by M.T.1) $T(n) = \Theta(n^{\log_b a}) = \underline{\Theta(n^2)}$

Example 8. $T(n) = 3T(\frac{n}{4}) + n \lg n$

$a = 3$, $b = 4$

$\log_b a = \log_4 3 \approx 0{,}793$

$f(n) = n \lg n = \Omega(n) = \Omega(n^{\log_4 3 + \varepsilon})$, for any $0 < \varepsilon \leq 1 - \log_4 3$; e.g. take $\varepsilon = 0{,}2$

$af(\frac{n}{b}) = 3 \cdot f(\frac{n}{4}) = 3 \cdot \frac{n}{4} \cdot \lg \frac{n}{4} = \frac{3}{4} n \lg n - \frac{3}{4} n \lg 4 = \frac{3}{4} n \lg n - \frac{3}{4} \cdot n \cdot 2 = \frac{3}{4} n \lg n - \frac{3}{2} n$

$cf(n) = cn \lg n$

Is there c such that $af(\frac{n}{b}) = \frac{3}{4} n \lg n - \frac{3}{2} n \leq cn \lg n = cf(n)$? (for sufficient large n)

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

$\Rightarrow$ M.T.3) $T(n) = \Theta(f(n)) = \Theta(n \lg n)$

Example 9. $T(n) = 2T(\frac{n}{2}) + n\lg n$

$a=2, b=2$

$\log_b a = \log_2 2 = 1 \Rightarrow n^{\log_b a} = n$

$f(n) = n\lg n$

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

It is rather: $f(n) = \Omega(n)$, so we cannot apply case 1) or case 2) of M.T., but 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 $n^{\log_b a} = n$.

$\frac{f(n)}{n} = \frac{n\lg n}{n} = \lg n$ which is smaller than n^ε for any $\varepsilon > 0$ (asymptotically)

$$\lim_{n\to\infty} \frac{\lg n}{n^\varepsilon} = \lim_{n\to\infty} \frac{\varepsilon \lg n}{\varepsilon n^\varepsilon} = \lim_{n\to\infty} \frac{1}{\varepsilon} \cdot \frac{\lg n^\varepsilon}{n^\varepsilon} = \frac{1}{\varepsilon} \lim_{x\to\infty} \frac{\lg x}{x} = 0$$

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

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

$af(\frac{n}{b}) = 2f(\frac{n}{2}) = 2\frac{n}{2}\cdot\lg\frac{n}{2} = n\lg n - n\lg 2 = n\lg n - n$

$cf(n) = cn\lg n$

$af(\frac{n}{b}) \leq cf(n) \iff$

$\Rightarrow n\lg n - n \leq cn\lg n$

$\Leftrightarrow n\lg n(1-c) \leq n$

$\Leftrightarrow \lg n(1-c) \leq 1$

However, for $c<1$, $1-c>0 \Rightarrow$ for large enough n ($n \geq 2^{\frac{1}{1-c}}$) inequality does NOT hold.

So, we cannot apply M.T.3)

What can we do then?

We can try some other method than M.T. (for example, we can try to solve it the same way we did before knowing M.T.), or we can apply some extensions of M.T. that have been developed.

Solution by "unrolling" $T(n)$ (as we did before)

$T(n) = 2T(\frac{n}{2}) + n\lg n$, assume that $n = 2^k$, $k = \lg n$

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

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

$= 4\left(2T(\frac{n}{8}) + \frac{n}{4}\lg\frac{n}{4}\right) + n\lg\frac{n}{2} + n\lg n$

$= 8T(\frac{n}{8}) + n\lg\frac{n}{4} + n\lg\frac{n}{2} + n\lg n$

$= \dots = nT(1) + n\lg 2 + n\lg 4 + \dots + n\lg\frac{n}{2} + n\lg n$

$= nT(1) + n\left(\lg 2 + \lg 4 + \dots + \lg\frac{n}{2} + \lg n\right)$

$= nT(1) + n\left(\lg 2^1 + \lg 2^2 + \dots + \lg 2^{k-1} + \lg 2^k\right)$

$= nT(1) + n(1 + 2 + \dots + k-1 + k)$

$= nT(1) + n\frac{k\cdot(k+1)}{2}$

$= nT(1) + n\cdot\frac{1}{2}\lg n(\lg n + 1)$

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

$T(n) = n\,\Theta(1) + \Theta(n\lg^2 n)$

$= \underline{\Theta(n\lg^2 n)}$

Extended Master Theorem (E.M.T.)

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

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

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

$\Rightarrow T(n) = \Theta(n^{\log_b a})$

2) If $f(n) = \Theta\left(n^{\log_b a}(\log_b n)^{-1}\right)$

$\Rightarrow T(n) = \Theta(n^{\log_b a}\log_b\log_b n)$

3) If $f(n) = \Theta(n^{\log_b a}(\log_b n)^\alpha)$ with $\alpha > -1$

$\Rightarrow T(n) = \Theta(n^{\log_b a}(\log_b n)^{\alpha+1})$

4) Same as M.T.3) , ie. if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$ and $\exists c < 1$ s.t. $af(\frac{n}{b}) \le cf(n)$ for large n, then $T(n) = \Theta(f(n))$

Solution to example 9. by applying E.M.T.

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

$a = 2, b = 2$

$n^{\log_b a} = n^{\log_2 2} = n^1 = n$

$f(n) = n\lg n = \Theta(n\lg n) = \Theta\left(n^{\log_b a} \cdot (\log_b n)^1\right) = \Theta\left(n^{\log_b a} \cdot (\log_b n)^{\alpha}\right)$, for $\alpha = 1$

$\alpha = 1 \Rightarrow$ we can apply E.M.T. 3)

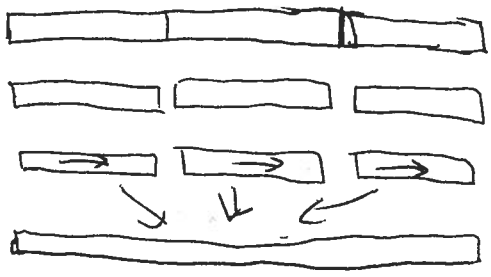
$\Rightarrow T(n) = \Theta(n^{\log_b a}(\log_b n)^{\alpha+1}) = \Theta(n(\lg n)^2) = \Theta(n\lg^2 n)$

Note that in this example $f(n)$ falls somewhere in between cases 2) and 3) of M.T. 1 and therefore E.M.T. 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 MergeSort is $\Theta(n\log n)$.

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



↓ split - $\Theta(1) = c_1^3$

↓ sort each subarray - $3T(\frac{n}{3})$

↓ merge - $\Theta(n) = c_2^3 \cdot n$

$T(n) = 3T(\frac{n}{3}) + f(n)$, where $f(n) = c_2^3 \cdot n + c_1^3$

$a = 3, b = 3$

$n^{\log_b a} = n^{\log_3 3} = n^1 = n$

$\Rightarrow f(n) = \Theta(n) = n^{\log_b a}$

$\Rightarrow$ M.T. 2) $T(n) = \Theta(n\log n)$

While both binary and ternary mergesort 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

$T^2(n) = 2T^2(\frac{n}{2}) + C_2^2 n + C_1^2$, where C_1^2 is cost of splitting, and $C_2^2 n$ cost of merging

$T^2(n) = 2(2T^2(\frac{n}{4}) + C_2^2 \frac{n}{2} + C_1^2) + C_2^2 n + C_1^2$

$= 4T^2(\frac{n}{4}) + C_2^2 n + C_2^2 n + 2C_1^2 + C_1^2$

$= \dots = nT^2(1) + \underbrace{C_2^2 n + \dots + C_2^2 n}_{\log_2 n \text{ times}} + C_1^2 + 2C_1^2 + \dots + 2^{k-1} C_1^2$ (assuming $n = 2^k$)

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

Ternary Merge Sort

$T^3(n) = 3T^3(\frac{n}{3}) + C_2^3 n + C_1^3$, where C_1^3 is cost of splitting, and $C_2^3 n$ is cost of merging.

$T^3(n) = 3(3T^3(\frac{n}{9}) + C_2^3 \frac{n}{3} + C_1^3) + C_2^3 n + C_1^3$

$= 9T^3(\frac{n}{9}) + C_2^3 n + C_2^3 n + 3C_1^3 + C_1^3$

$= \dots = nT^3(1) + \underbrace{C_2^3 n + \dots + C_2^3 n}_{\log_3 n \text{ times}} + C_1^3 + 3C_1^3 + \dots + 3^{k-1} C_1^3$ (assuming $n = 3^k$)

$= nT^3(1) + C_2^3 n \log_3 n + \frac{n-1}{2} C_1^3$

$= nT^3(1) + C_2^3 n \log_3 2 \log_2 n + \frac{n-1}{2} C_1^3$

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

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

$T^3(n) \approx C_2^3 \log_3 2 \, n \log_2 n$

$\Rightarrow$ we need to compare C_2^2 and $C_2^3 \log_3 2$

$\log_3 2 \approx 0{,}63 \Rightarrow$ compare C_2^2 and $0{,}63 C_2^3$

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 $C_2^3 > C_2^2$, but we don't know exact ratios because there are other operations involved and we need to know cost model of the programming lang