DNIDE-AND-CONQUER ALGORITHMS

Date

Let

$T(n)$ be the running time of a recursive algorithm. When n is small, say, $n << c$ (some constant c)

then,

$T(n) = \Theta(1)$ ← base case

Suppose,

our division of the problem into sub-problems makes 'a' subproblems, each of size 'n/b'

then, it takes $T\left(\frac{n}{b}\right)$ time to solve one sub-problem of size $\left(\frac{n}{b}\right)$ and $aT\left(\frac{n}{b}\right)$ to solve 'a' subproblems

If $D(n)$ is the time needed (cost) to divide and $C(n)$ is the time needed to combine, the resulting recurrence is given as:

$$T(n) = \begin{cases} \Theta(1), & n << c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n), & \text{otherwise} \end{cases}$$

Example:

Given $T(n) = \begin{cases} 2, & n = 2 \\ 2T\left(\frac{n}{2}\right) + n; & n = 2^k (k>1) \end{cases}$

(n is in exact powers of 2)

Show that $T(n) = n \lg n$

Let's check for the base-case: $n = 2$

$$T(n=2) = 2 \cdot \lg_2 2 = 2(1) = 2$$

$\because$ n is in exact powers of 2 and $T(n) = 2T\left(\frac{n}{2}\right) + n$ [Given]

the next term is $T(2n) = 2T\left(\frac{2n}{2}\right) + 2n$

$$\Rightarrow T(2n) = 2T(n) + 2n = 2(n \lg n) + 2n$$

($n \lg n$: given hypothesis inductive)

RC

$\Rightarrow T(2n) = 2\,(n \lg n) + 2n$

[Recall: $\lg n = \lg \frac{2n}{2} \Rightarrow \lg n = \lg 2n - \lg 2 = \lg 2n - 1$]

So,

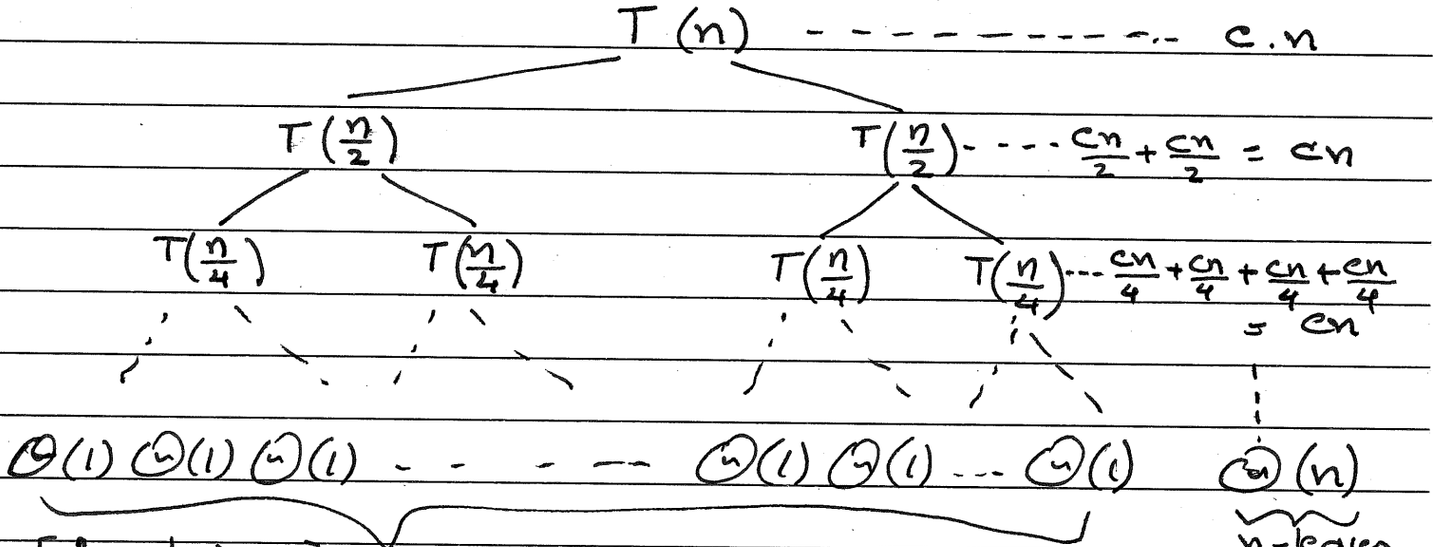
$$T(2n) = 2n\,[\lg 2n - 1] + 2n$$

$$= 2n \lg 2n - 2n + 2n$$

$$\therefore \boxed{T(2n) = 2n \lg 2n}$$

The recurrence we just solved by induction is that of MergeSort [from DSA]

Let's draw the Recurrence Tree :-



$[2^h = 2^{\lg_2 n} = n]$ n leaves :

So, the total cost of MergeSort on an input of size 'n' is the Sum of work done by intermediate nodes and the work done by the leaves in the recurrence tree.

ie, $T(n) = 2\,T\left(\frac{n}{2}\right) + c.n$

- 2 → # of subproblems → 'a' → branching factor
- $\frac{n}{2}$ → dividing factor (RC)
- $c.n$ → cost to merge → combine

Assuming (w log) that n is in the exact powers of 2 (here '2' being the dividing factor 'b'

then, it would take $\lg_2 n + 1$ steps to reach the terminal (leaves) nodes.

The total cost at each level of the recurrence tree is $c.n$.

Hence,

$$T(n) = cn\left[\lg_2 n + 1\right] = cn\lg n + cn$$

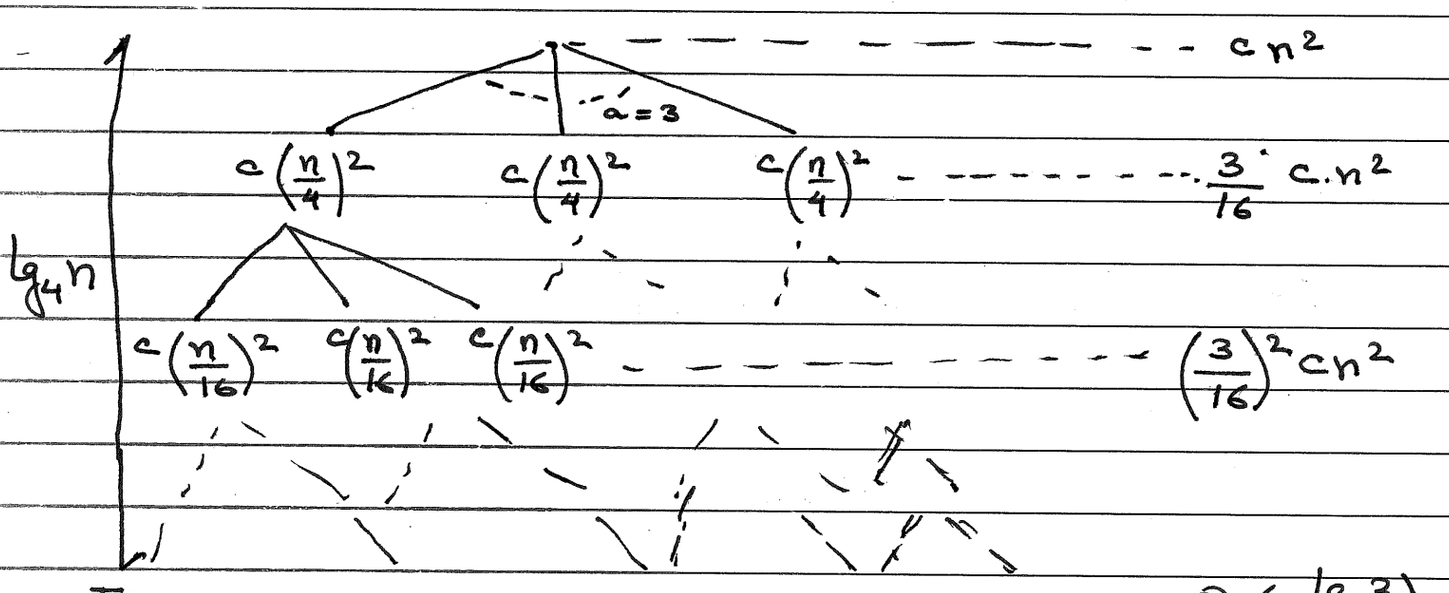
or $\boxed{T(n) = \Theta(n \lg n)}$

Source: Cormen et al. (2022) [CLRS]: Let's take another example

Let

$$T(n) = 3\,T\left(\frac{n}{4}\right) + \underbrace{c.n^2}_{f(n)}$$

['a': branching factor is 3; 'b': dividing factor is 4]



# of leaves is $3^{\lg_4 n} = n^{\lg_4 3}$

height h: $\lg_4 n$; # of levels: $\lg_4 n + 1$.

RC

Date

∵ Each subproblem size decreases by a factor of 4 ('4'), we arrive at the base-case in $\log_4 n$ steps (i.e., with $0, 1, \ldots, \log_4 n$ levels).

So, $T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} + \Theta\left(n^{\log_4 3}\right)$

$\Theta(n^{\epsilon})$ $[\epsilon > 0]$?

[Is, the sequence increasing or decreasing?]

$$\Rightarrow T(n) = \sum_{j=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^j cn^2 + \Theta\left(n^{\log_4 3}\right)$$

Work done at intermediate nodes — Cost at leaves's level

i.e.,

$$T(n) = \frac{\left(\frac{3}{16}\right)^{\log_4 n} - 1}{\left(\frac{3}{16}\right) - 1} cn^2 + \Theta\left(n^{\log_4 3}\right) \quad \left[\because \frac{x^n - 1}{x - 1} \text{ for finite seq.}\right]$$

Now,

$$T(n) = \sum_{j=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^j cn^2 + \Theta\left(n^{\log_4 3}\right)$$

$$< \sum_{j} \left(\frac{3}{16}\right)^j cn^2 + \Theta\left(n^{\log_4 3}\right)$$

$$= \frac{1}{1 - \frac{3}{16}} cn^2 + \Theta\left(n^{\log_4 3}\right)$$

or

$$\boxed{T(n) = \Theta(n^2)}$$

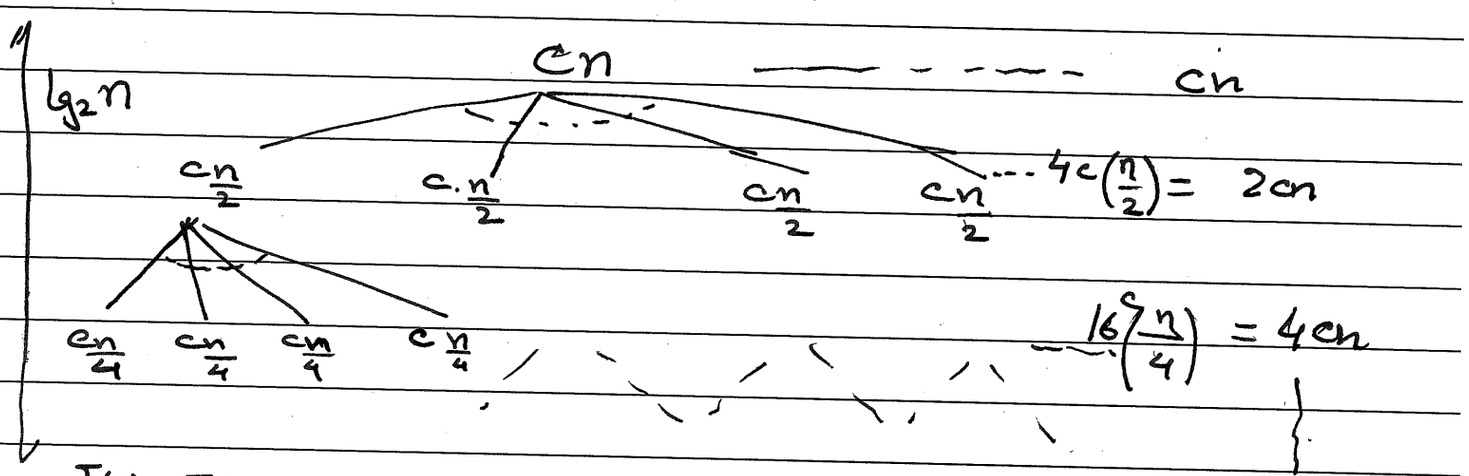
RC

Date

So, one final example, before we generalize them in master theorem:

$$T(n) = 4\,T\left(\frac{n}{2}\right) + O(n) \qquad \left[\begin{matrix} a = 4 \\ b = 2 \end{matrix}\right]$$

Let's sketch the recurrence tree:



So, how many leaves: $h: \lg_2 n$

$$\#\text{ of leaves: } 4^{\lg_2 n} = n^{\lg_2 4} = n^2 \quad \cdots \quad \Theta(n^2)$$

Let, $T(n)$ be the total work done:

$$T(n) = \sum_{j=0}^{\lg_2 n - 1} (2)^j cn + \Theta(n^2)$$

$cn + 2cn + 4cn + \cdots$ ($\lg_2 n$ terms)

$$= cn \sum_{j=0}^{\lg_2 n - 1} 2^j + \Theta(n^2)$$

$$= cn\,\frac{2^{\lg_2 n - 1} - 1}{2 - 1} + \Theta(n^2)$$

$$\therefore \boxed{T(n) = \Theta(n^2)}$$

And now, we're ready for the Proof of Master Theorem!

RC

Proof of Master Theorem (CLRS)

Let's assume that [w.l.o.g] n is in exact powers of the dividing factor ('b'). [$b > 1$] — why?

The general recursion is given as:

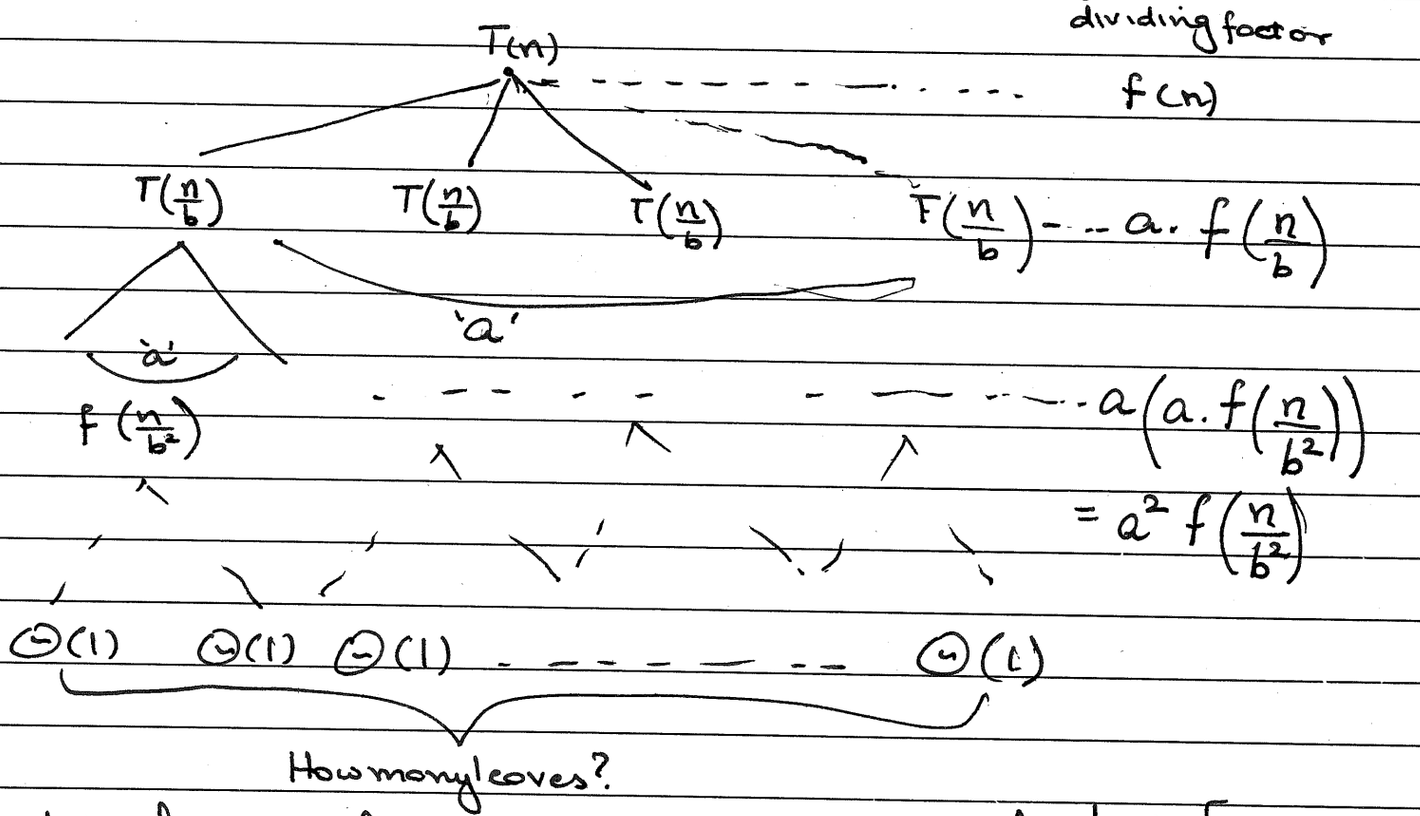
$$T(n) = \begin{cases} \Theta(1); & n << c \ [n = 1, i = 0] \\ aT\left(\frac{n}{b}\right) + f(n); & n = b^i \ [i > 0] \end{cases}$$

$\hookrightarrow D(n) + C(n)$

Let's sketch its recursion tree:

The root of the tree has a cost $f(n)$, has 'a' children (subproblems) with a cost of $f\left(\frac{n}{b}\right)$ each.

↳ dividing factor



Let h be the height of the recursion tree; $h = \log_b n$ [an 'a'-ary tree]

# of leaves : $a^{\log_b n} = n^{\log_b a}$

Work done at leaves: $\Theta(n^{\log_b a})$

The total work done is the sum of work done by the internal nodes and the work done by leaves.

i.e., $$T(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right) + \Theta(n^{\log_b a})$$

The three (03) cases of MasterThm are about how cost is distributed across levels.

Proof of Case I: Cost dominated by leaves

or

$$f(n) = O(n^{\log_b a - \epsilon})$$

[$f(n) = \frac{O(n^{\log_b a})}{O(n^\epsilon)}$ ← polynomially larger than $f(n)$]

i.e., we're comparing $f(n)$ with $O(n^{\log_b a})$; the larger of the two will dominate the asymptotic cost of the recurrence.

$$f(n) = O(n^{\log_b a - \epsilon}), \epsilon > 0; \quad T(n) = \Theta(n^{\log_b a})$$

Now, at the (some) j^{th} level of the recurrence tree:

$$f\left(\frac{n}{b^j}\right) = O\left(\left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}\right)$$

Now, work done by all nodes at the j^{th} level is:

$$\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon} = n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} \left(\frac{a b^\epsilon}{b^{\log_b a}}\right)^j$$

($\frac{a b^\epsilon}{b^{\log_b a}} = a$; a constant)

$$= n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} (b^\epsilon)^j = n^{\log_b a - \epsilon} \cdot \frac{b^{\epsilon \log_b n} - 1}{b^\epsilon - 1}$$

$$= n^{\log_b a - \epsilon} \left[\frac{n^\epsilon - 1}{b^\epsilon - 1}\right] \quad \because b \text{ and } \epsilon \text{ are constants}$$

$$\left[\sum_{i=1}^{n} x^j = \frac{x^n - 1}{x - 1}\right]$$

$$n^{\log_b a - \epsilon} \, O(n^\epsilon) = O(n^{\log_b a})$$

work done by the leaves.

RC

Case II : Assumes $f(n) = \Theta(n^{\log_b a})$

Then, at the j^{th} level of the recurrence tree,
work done at a node is $f\left(\frac{n}{b^j}\right) = \Theta\left(\frac{n}{b^j}\right)^{\log_b a}$

ab the total work at the j^{th} level:

$$\Theta\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}\right) = n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \left(\frac{a}{\underbrace{b^{\log_b a}}_{=a}}\right)^j$$

No geometric progression ; it's all the same!

$$= n^{\log_b a} \log_b n = \Theta\left(n^{\log_b a} \log_b n\right)$$ [try MergeSort!]

Case III : When the work done at root is dominant!

i.e.

$$f(n) = O\left(n^{\log_b a + \epsilon}\right), \quad \epsilon > 0$$

Then, we have a decreasing geometric sequence

work done by internal nodes is $\sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$

$$\leq \sum_{j=0}^{\log_b n - 1} c^j f(n) + \underbrace{O(1)}_{\text{some constant cost}}$$

$$\leq \sum_{j=0}^{\infty} f(n) c^j + O(1) = f(n) \sum_j c^j + O(1)$$

$$= f(n) \left(\frac{1}{1-c}\right) + O(1)$$

and so,

$$\boxed{T(n) = \Theta\left(n^{\log_b a + \epsilon}\right)}$$

$f(n)$.