Divide - and - conquer

Recurrences are one way to express the running time of recursive algorithms and in particular of D-a-C ones.
Recurrences describe the running time of an algo in terms of its smaller inputs.
For mergesort, the running time can be defined as:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ \boxed{2}\, T\left(\boxed{\frac{n}{2}}\right) + \Theta(n) & \text{otherwise} \end{cases}$$

\# of subproblems — size of the input for every subproblem

The input instance can be divided into subproblems of different sizes
for instance, if we have a 1/3 and a 2/3 split we can write:

$$T(n) = T\left(\frac{2n}{3}\right) + T\left(\frac{n}{3}\right) + f(n)$$

Or in the case of linear search we do not split the input, but we work on consequently smaller inputs : $T(n) = T(n-1) + \Theta(1)$

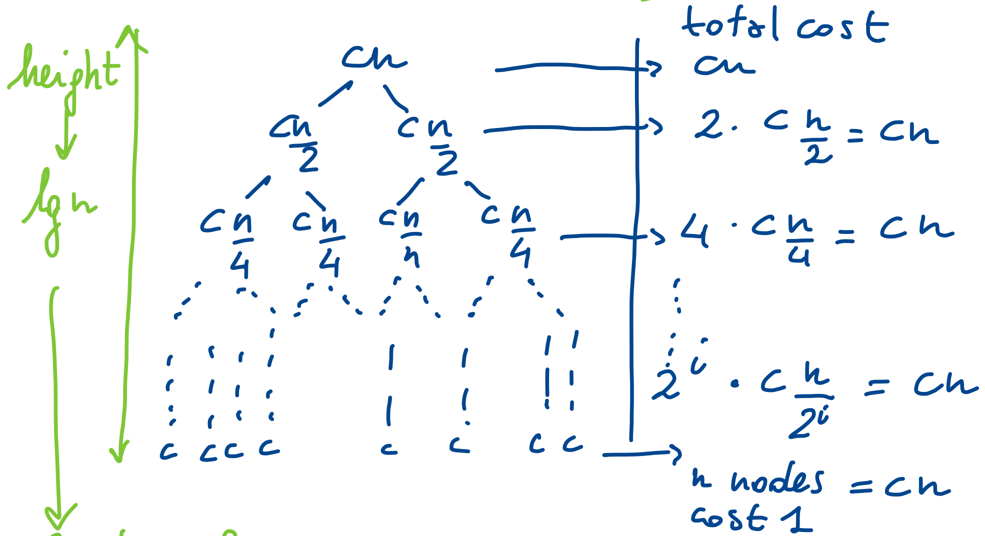
We will see 3 ways to solve recurrences :
1. Recurrence trees
2. Substitution
3. Master method.

Merge sort running time by using recurrence trees

1. We assume, without loss of generality, that the input size is a power of 2

Divide : we compute the middle of an array : $\Theta(1) = D(n)$
Conquer: we recursively solve 2 subproblems, each of size $n/2 \Rightarrow 2T(n/2)$
Combine : the merge operation $C(n) = \Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) + \underbrace{\Theta(1)}_{\text{we can ignore this}}, & \text{otherwise} \end{cases}$$



levels = height + 1 → A tree with one node has 1 level and height = ∅

The numbers of levels of the recursion is $\lg n + 1$ where n is the number of leaves (size of the input)

WHY?

Induction:

Base $n = 1$ then we have 1 level $\Rightarrow \lg 1 + 1 = 1$

Inductive step $n = 2^i \Rightarrow$ levels $= \log 2^i + 1 \Rightarrow i \lg 2 + 1 = i + 1$

→ since n is a power of 2, the next level to consider is 2^{i+1}

so: $\lg 2^{i+1} + 1$

We know that a tree with 2^{i+1} leaves has just one more level than a tree with 2^i leaves so since levels$(2^i) = i+1$, levels$(2^{i+1}) = i+2$

so $\lg 2^{i+1} + 1 = \lg 2 + i \lg 2 + 1 = (i+1)\lg 2 + 1 = i + 2$

Then, a binary tree with n leaves, has $\lg n + 1$ levels each taking cn work

$\Rightarrow T(n) = \Theta((\lg n + 1)cn) \Rightarrow \Theta(n \lg n + n) = \Theta(n \lg n)$

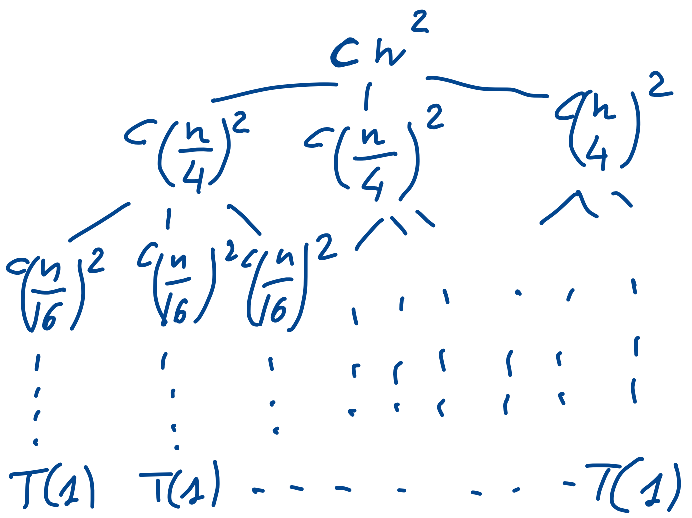
we ignore constants

Recurrence trees

Solve the following recursion:

$$T(n) = 3T(\lfloor \tfrac{n}{4} \rfloor) + \Theta(n^2)$$

1. With a tolerable amount of sloppiness we ignore floors and ceilings
2. The input is divided, recursively, into 3 subproblems with size $n/4$. Each level requires cn^2 work



The subproblem size at depth i is $\frac{n}{4^i}$ → when do we reach size $n = 1$?

$\frac{n}{4^i} = 1 \qquad n = 4^i$

$\Rightarrow \boxed{i = \log_4 n}$ → height

↳ total # levels $= \log_4 n - 1$

What is the cost of each level?

Each level has 3 times the nodes of the level above, so at level i we have 3^i nodes. The subproblem size reduces by a factor 4 for each level, so each node for $i = 0, 1, 2, \dots, \log_4 n - 1$ has a cost $c\left(\frac{n}{4^i}\right)^2$ so given that at level i we have 3^i nodes, the cost for each level is $3^i c\left(\frac{n}{4^i}\right)^2$

$$\Rightarrow \left(\frac{3}{16}\right)^i c n^2$$

At the leaves level we have $3^{\log_4 n}$ nodes each one requiring $\Theta(1)$ work $\Rightarrow n^{\log_4 3}$

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

We can approximate to:

$$T(n) = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i c n^2 + \Theta\left(n^{\log_4 3}\right) < \sum_{i=0}^{+\infty} \left(\frac{3}{16}\right)^i c n^2 + \Theta\left(n^{\log_4 3}\right)$$

Remember that $\sum_{k=0}^{+\infty} x^k = \frac{1}{1-x}$ if $|x| < 1$ $\Rightarrow \frac{3}{16} = x < 1$

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

$$= \frac{16}{13} c n^2 + \Theta\left(n^{\log_4 3}\right) \Rightarrow O(n^2)$$

Valid for big-oh / not big-Theta

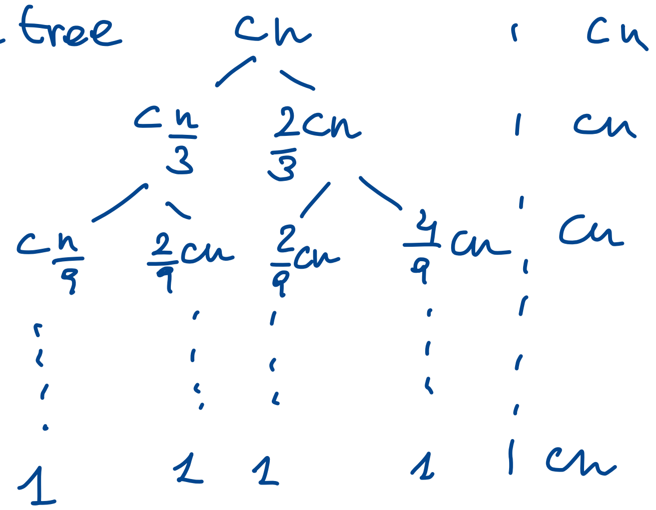
In general, given a recurrence $T(n) = aT\left(\frac{n}{b}\right) + \Theta(1)$

$$h = \log_b n \qquad \# \text{leaves} = n^{\log_b a}$$

Exercise: $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2}{3} n\right) + O(n)$ → What's the complexity?

Exam alert!

We build the recursion tree



We have to determine the height of the tree, we consider the longest path from root to leaves $\Rightarrow T(\frac{2}{3}n)$

$$\Rightarrow \frac{1}{b} = \frac{2}{3} \Rightarrow b = \frac{3}{2}$$

$h = \log_{3/2} n$ each level costs cn, so the overall cost is $O(n \lg n)$

How many leaves does the tree have?
$\#$ leaves $= n^{\log_{3/2} 2}$ if we'd have a complete binary tree
$\Rightarrow w(n \lg n)$ for a given constant w.

We have to check if $O(n \lg n)$ is a reasonable boundary given that we do not have a complete binary tree and thus we have to perform less work.

We have to show that $T(n) \leq d\, n \lg n$, where d is a positive constant

$$T(n) \leq T(n/3) + T(\tfrac{2}{3}n) + cn \leq d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn$$

$$= (d(n/3)\lg n - d(n/3)\lg 3) + (d(\tfrac{2n}{3})\lg n - d(\tfrac{2n}{3})\lg(\tfrac{3}{2})) + cn$$

$$\vdots$$

$$= d\, n \lg n - dn(\lg 3 - 2/3) + cn$$

$$\leq d\, n \lg n$$

so, as long as $d \geq \dfrac{c}{(\lg 3 - 2/3)}$, $O(n \lg n)$ is an admissible upper bound