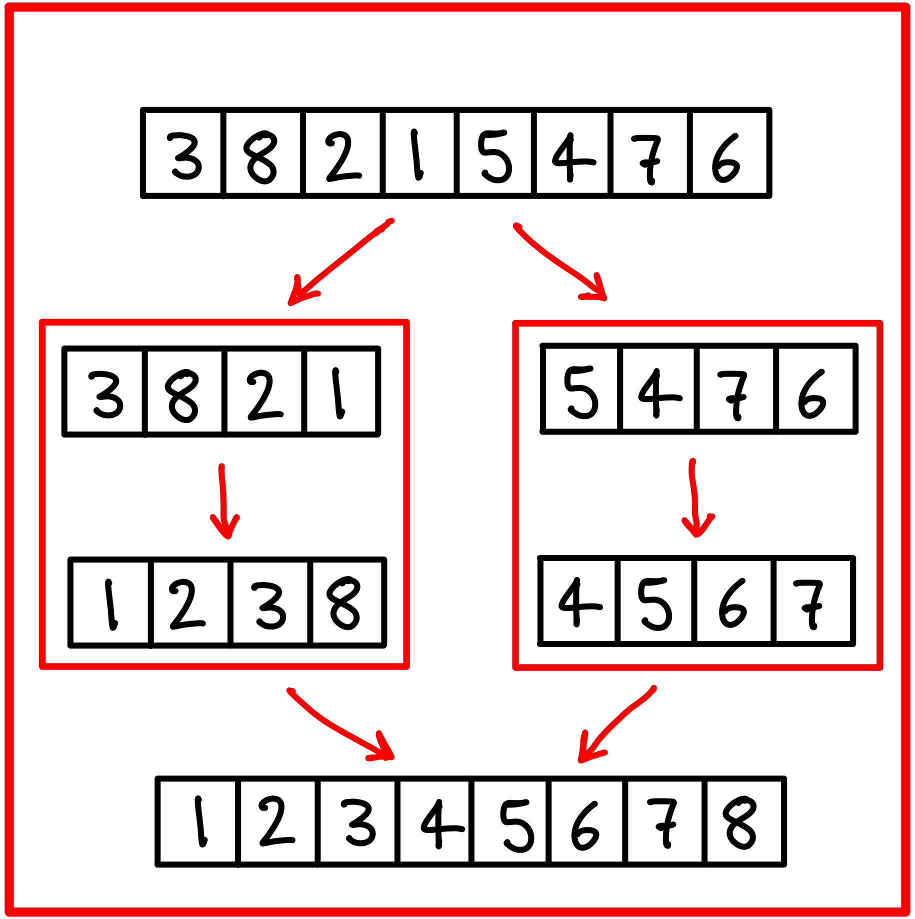
MERGE SORT, Divide & Conquer, dealing with recurrence relations

Divide problem into 2 smaller instances

Conquer (= solve) the smaller problems
(Mergesort)

Combine (= merge) the solutions



Merging 2 sorted arrays:

$\boxed{\Theta(n) \text{ time}}$

Smallest element is at leftmost position of A or B.

In general: copy to output, increment, smallest remaining element found with 1 comparison.

A:

1	2	3	5	9	14	16	20

B:

4	6	10	11	15	19	25	31

Increment index of A

Increment A

Increment A

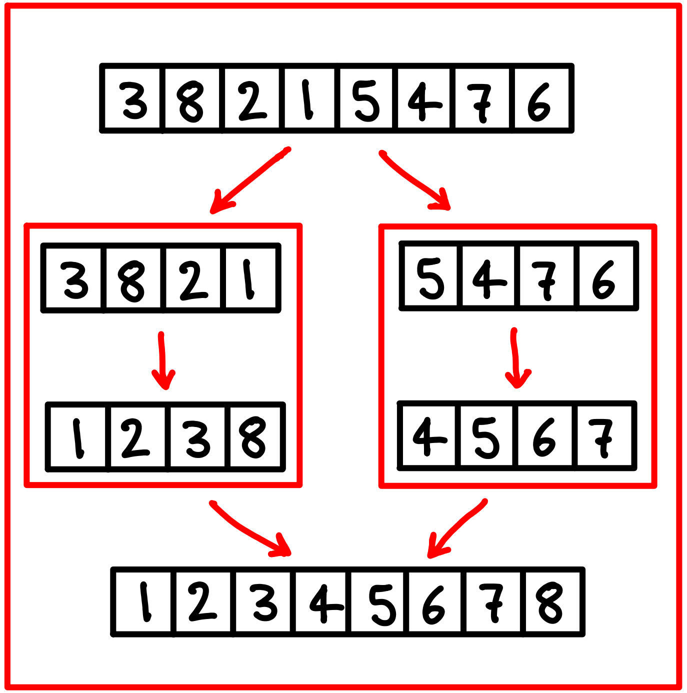
Increment B

etc

output:

1	2	3	4												

Mergesort time for n elements: $T(n)$



1) Divide $\Theta(1)$

2) Conquer $\Theta(1) + 2\cdot T(\frac{n}{2})$

3) Merge $\Theta(n)$

$$\boxed{T(n) = 2\cdot T(\tfrac{n}{2}) + \Theta(n)}$$

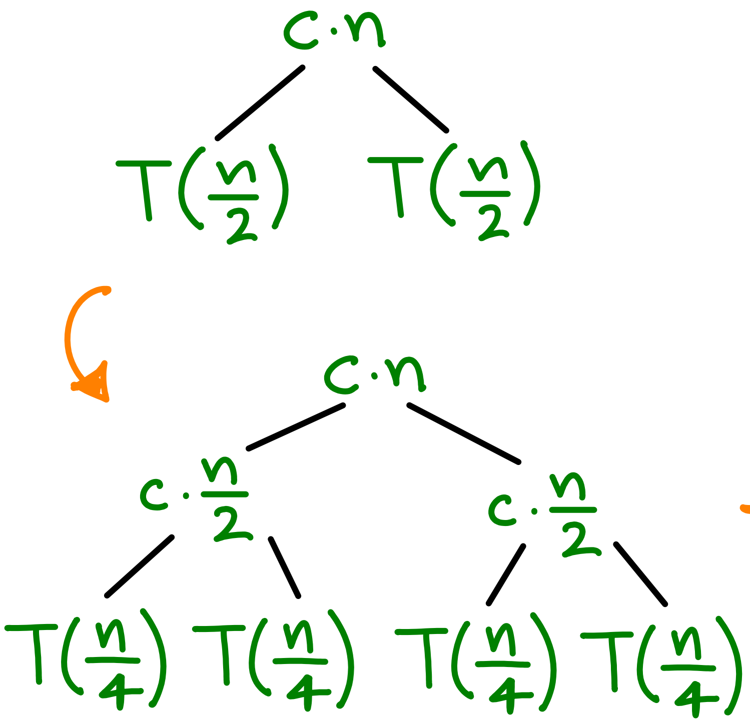
$$T(1) = \Theta(1)$$

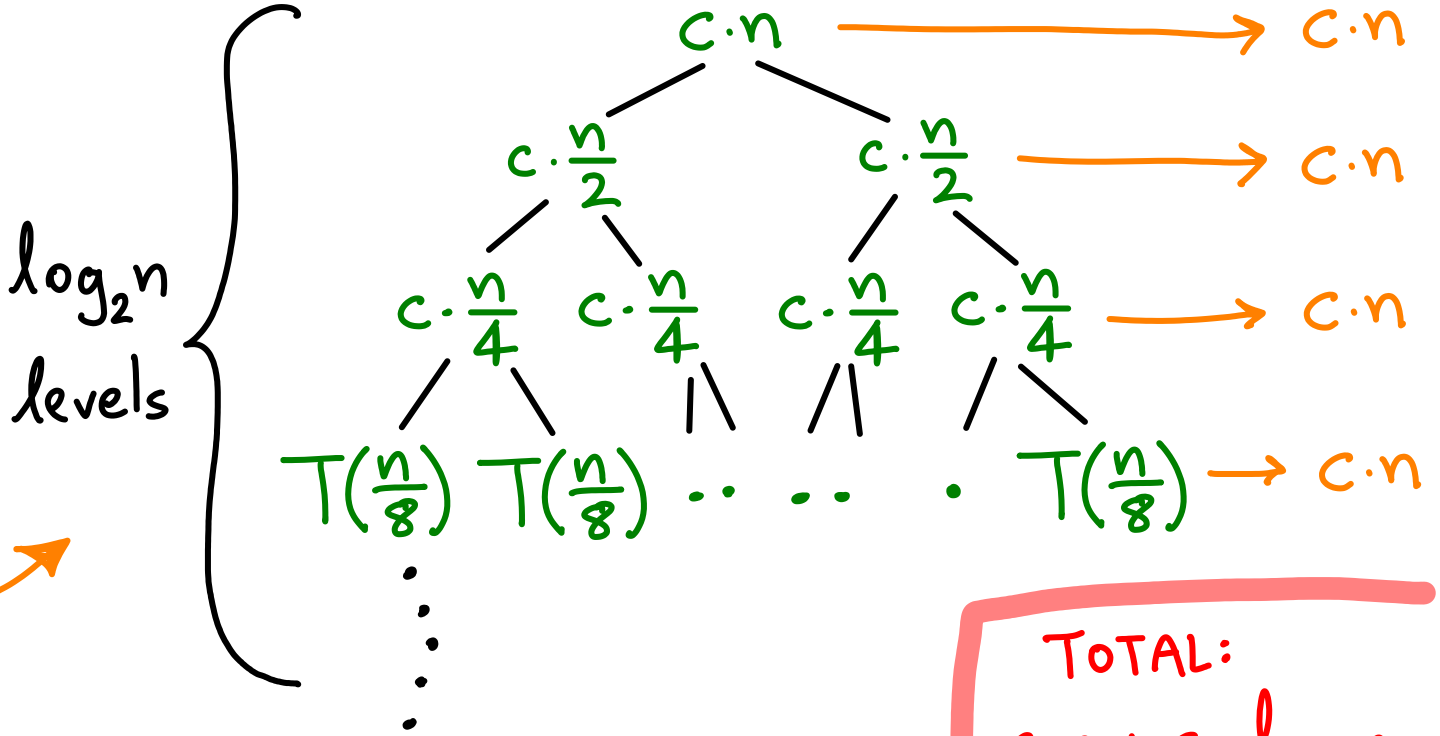
Actually $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$ → easy to deal with

How to solve $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \underbrace{\Theta(n)}$ $\quad T(1) = \Theta(1)$

$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + c \cdot n$ ← must do this → $= c_2$

Recursion tree:





n leaves: c_2 c_2 c_2 ...

$$\boxed{\begin{array}{l}\text{TOTAL:}\\ c_2 n + c n \log_2 n\\ = \Theta(n \log n)\end{array}}$$

The recursion tree method relies on noticing a pattern and for it to be formal one must prove that the pattern holds not just for a few levels.

If that's not easy to do, use substitution and induction...

How to solve $T(n) = 2\cdot T(\frac{n}{2}) + \underbrace{\Theta(n)}$ by substitution (induction)

$T(n) = 2\cdot T(\frac{n}{2}) + c\cdot n$ ← must do this

part of input → no control

0) You need to have a guess for the answer. we control this

↳ focus on upper bound: $T(n) \leq d\cdot n\log n$? → $O(n\log n)$

1) Inductive hypothesis: for all $k<n$, $T(k) \leq d\cdot k\log k$

2) Substitute: $T(n) \leq 2\cdot d\frac{n}{2}\log\frac{n}{2} + cn$ (using $k=\frac{n}{2}$)

3) Algebra:

$$\begin{cases} = dn\log n - dn\log 2 + cn \\ = dn\log n - (dn - cn) \\ \leq dn\log n \ \ldots \text{ if } d \geq c \end{cases}$$

: desired form − leftovers

$\square$ (base case omitted)

More observations:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + c \cdot n$$

- For an upper bound, you <u>must</u> get the desired form or less.

  (exactly: with the same leading constant)

- Like any inductive proof,

  if it doesn't work, that doesn't imply it's not true.

- For mergesort specifically, the constant c comes from the merge step

  ...and this ends up as the leading constant. $T(n) \le c\, n \log n$

(Speedup of mergesort is directly proportional to speedup of merge)

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

What should we guess?

$T(1) = 1$

$$T(2) = 4T(1) + 2 = 4 \cdot 1 + 2 = 6$$

$$T(4) = 4T(2) + 4 = 4 \cdot 6 + 4 = 28$$

$$T(8) = 4T(4) + 8 = 4 \cdot 28 + 8 = 120$$

$$T(16) = 4T(8) + 16 = 4 \cdot 120 + 16 = 496$$

$$T(32) = 4T(16) + 32 = 4 \cdot 496 + 32 = 2016$$

: starts looking like $2n^2$

2nd term less and less significant

$T(n) = 4T(\frac{n}{2}) + n$ $\quad T(1) = 1$ $\quad$ What should we guess?

It looked quadratic. Not really convincing though. Let's be cautious.

Try $O(n^3)$: $T(n) \overset{?}{\leq} cn^3$

Hypothesis: $T(k) \leq ck^3$ for $k < n$

Substitute: $T(n) \leq 4 \cdot c \cdot \left(\frac{n}{2}\right)^3 + n$

Algebra:
$$= \frac{1}{2}cn^3 + n$$
$$= \underbrace{cn^3}_{\text{desired form}} - \frac{1}{2}cn^3 + n = cn^3 - \overbrace{\underbrace{\left(\frac{1}{2}cn^3 - n\right)}_{\geq 0 \text{ if } c \geq 2 \text{ for all } n \geq 0}}^{\geq 0 \text{ if } c \geq 1 \text{ \& } n \geq 2} \rightarrow T(n) \leq cn^3$$

but now you must handle a larger base case

Base case: $T(1) = 1 \leq c \cdot 1^3$ ✓ $\square$

We wanted to show $T(n) = 4T(\frac{n}{2}) + n = O(n^3)$.

Why use hypothesis $T(k) \leq ck^3$ instead of $T(k) = O(k^3)$?

e.g., $T(n) = 4T(\frac{n}{2}) + n$

$$= 4 \cdot O((\tfrac{n}{2})^3) + n = 4 \cdot O(\tfrac{1}{8}n^3) + n = O(n^3)$$

Let's show $T(n) = n$ is $O(1)$

Base case: $T(1) = O(1)$ ✓

Hypothesis: $T(k) = O(1)$ for all $k < n$

$\hookrightarrow T(n-1) = n-1 = O(1)$

$T(n) = n = n-1+1 = O(1) + 1 = O(1)$

DON'T USE BIG-O DURING INDUCTION

$T(n) = 4T\left(\frac{n}{2}\right) + n$ / $T(1) = 1$ / We proved $O(n^3)$. Let's try $O(n^2)$

$$T(n) \overset{?}{\leq} cn^2$$

Hypothesis: $T(k) \leq ck^2$ for $k < n$

Substitute: $T(n) \leq 4 \cdot c \cdot \left(\frac{n}{2}\right)^2 + n$

Algebra: $= cn^2 + n$ → failed to get $\leq$ desired form

$$= cn^2 + \frac{1}{n}n^2 = \left(c + \frac{1}{n}\right) \cdot n^2$$

so close to cn^2
but not good enough

$T(n) = 4T\left(\frac{n}{2}\right) + n$ / $T(1) = 1$ / We proved $O(n^3)$. We want $O(n^2)$

Hypothesis: $T(k) \leq ck^2$ failed. New hypothesis: $T(k) \leq ck^2 - dk$

Making the hypothesis stronger often works. See my notes on induction.

Substitute: $T(n) \leq 4\cdot\left(c\left(\frac{n}{2}\right)^2 - d\frac{n}{2}\right) + n$

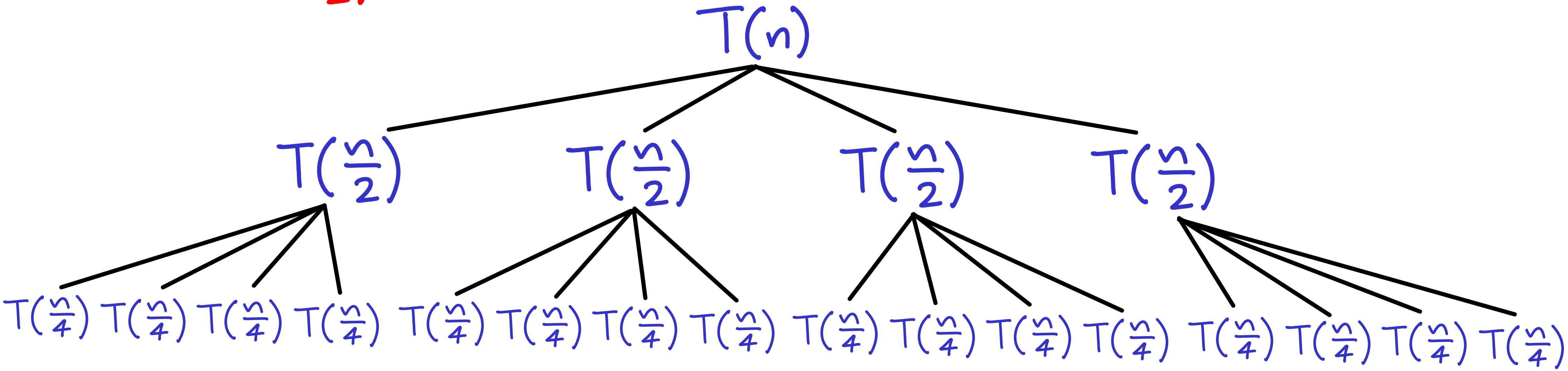
Algebra: $= cn^2 - 2dn + n$

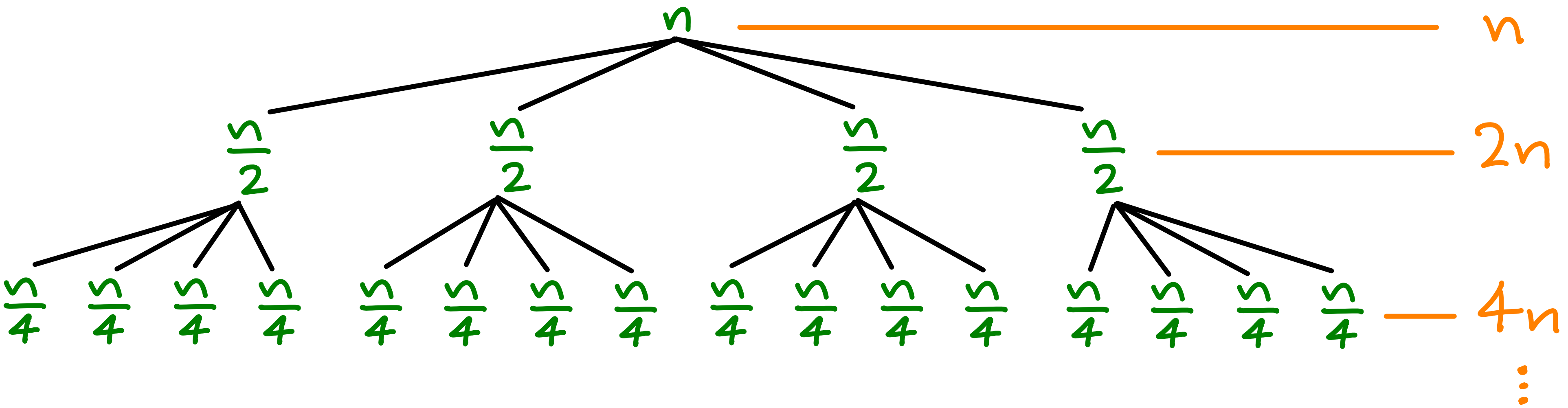
$$= \underbrace{cn^2 - dn}_{\text{desired form}} - dn + n = cn^2 - dn - \underbrace{(dn - n)}_{\geq 0 \text{ if } d \geq 1}$$

So can c be anything? No.

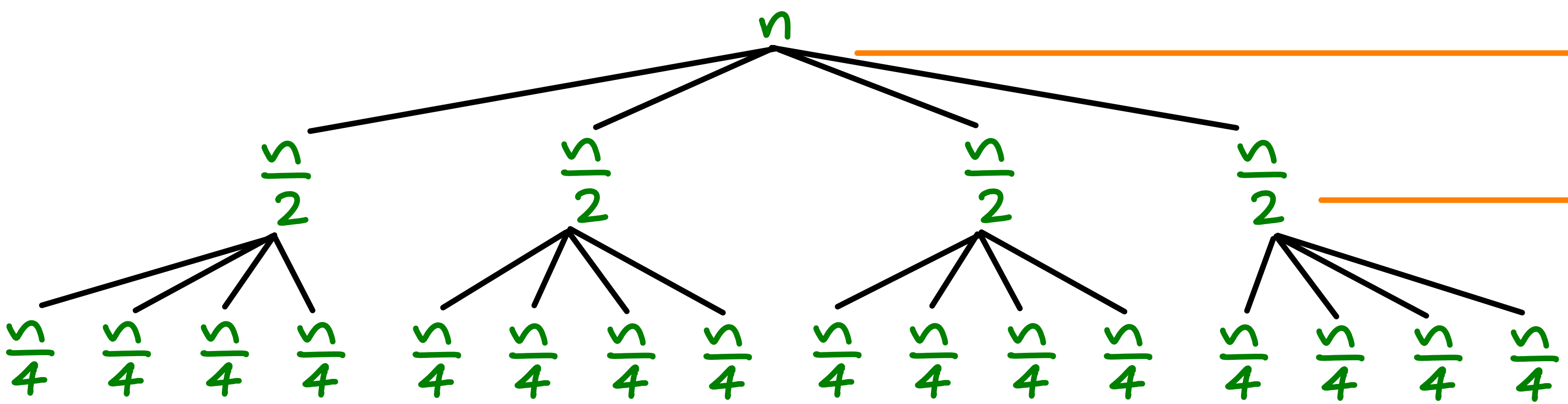
Base case: $T(1) = 1 \leq c\cdot 1^2 - d\cdot 1$ if $c > d$

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





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



level 0 n

level 1 $2n$

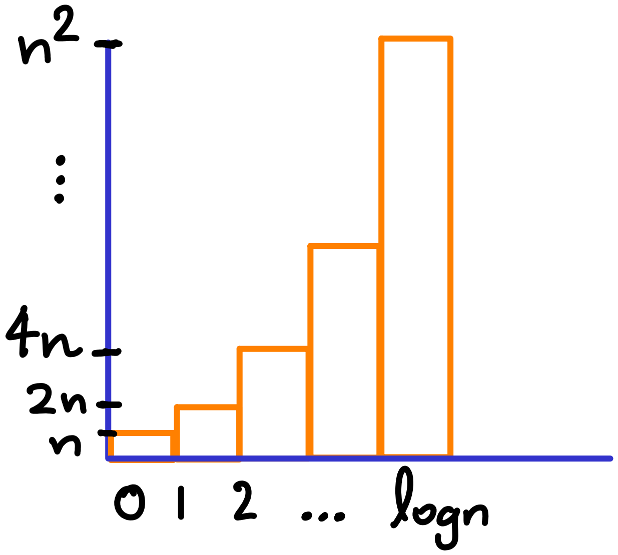
level 2 $4n$

⋮

level k $2^k \cdot n$

Geometric series

Sum $= 2n^2$

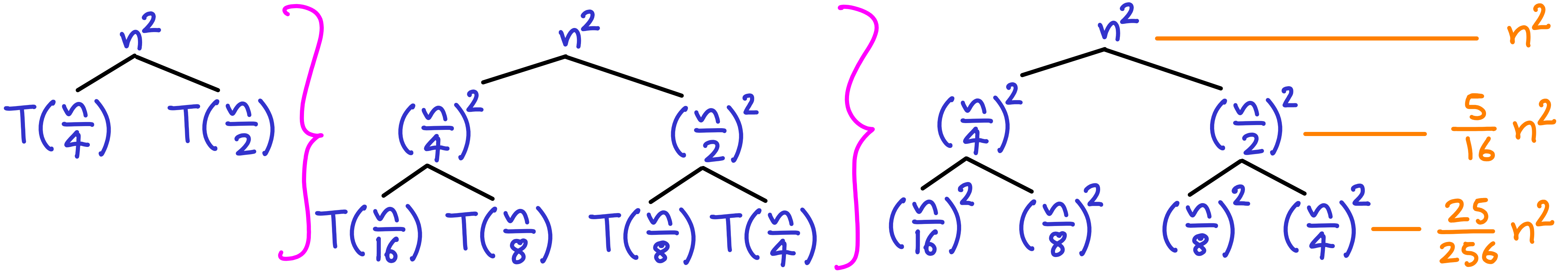


leaf level $\sim \log_2 n$ $2^{\log_2 n} \cdot n$

$= n^2$

SUM $= O(n^2)$

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



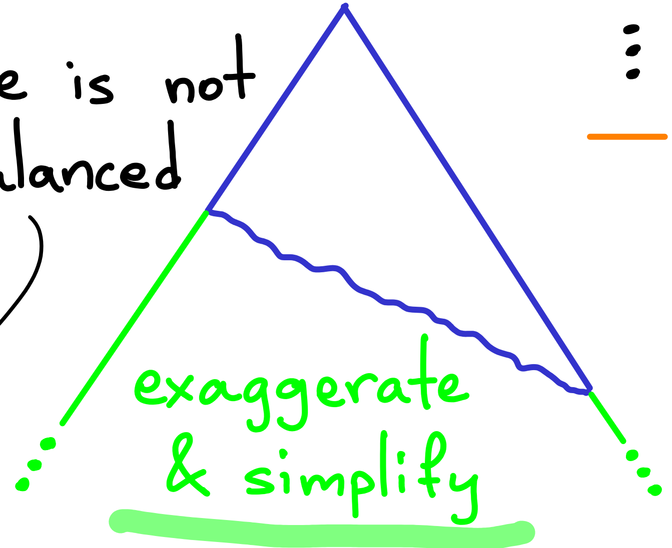
$\vdots$

$\left(\frac{5}{16}\right)^k n^2$

$$T(n) \le n^2 \cdot \left[1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots + \left(\frac{5}{16}\right)^\infty\right]$$

$$< n^2 \cdot \left[1 + \frac{1}{2} + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \cdots\right] = 2n^2$$

tree is not balanced



pretend the pattern holds forever