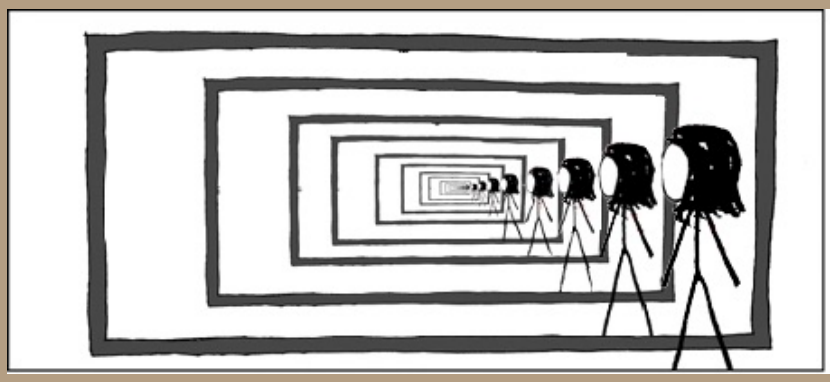
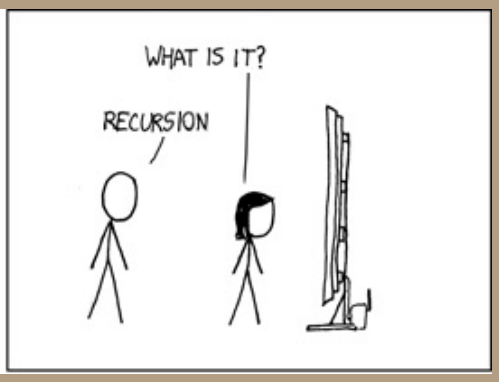
Algorithms





Recursion (part 2)

Recap:

- HW0 due tonight - in a group!
- Reading due tomorrow (5 comments now!)
- Next HW (over recursion) is posted, as well as next ~2 weeks of reading
  - ↳ due next Wed.
  - 3 question HW1 / Groups up

- Questions?
- Set office hours by Friday

Today: Recurrences!

(They really do have a purpose!)

Most "easy" asymptotic analysis in prior classes was probably loops.

→ loops. ⇝ sums [for $i \leftarrow 1$ to n {[code: #ops]}] $\sum_{i=1}^{n}$ (#ops)

Problem: Recursion frames things another way!

Ex

let $M(n)$ = runtime on array of length n

```
MERGESORT(A[1..n]):
  if n > 1
    m ← ⌊n/2⌋
    MERGESORT(A[1..m])          ⟨⟨Recurse!⟩⟩
    MERGESORT(A[m+1..n])        ⟨⟨Recurse!⟩⟩
    MERGE(A[1..n], m)
```

```
MERGE(A[1..n], m):
  i ← 1; j ← m+1                          ← O(1)
  for k ← 1 to n
    if j > n
      B[k] ← A[i]; i ← i+1
    else if i > m
      B[k] ← A[j]; j ← j+1
    else if A[i] < A[j]
      B[k] ← A[i]; i ← i+1
    else
      B[k] ← A[j]; j ← j+1
  for k ← 1 to n
    A[k] ← B[k]
```

Figure 1.6. Mergesort

O(1) + ?? + O(n)

$M(0) = M(1) = O(1)$

$M(n) = O(1) + O(n) + M(m) + M(n-m)$

$M(0) = M(1) = O(1)$

$M(n) = O(1) + O(n) + M(m) + M(n-m)$

where $m = \lfloor \frac{n}{2} \rfloor$

$\Rightarrow$ Mergesort runtime is solution to

$$M(n) = M\left(\lfloor \tfrac{n}{2} \rfloor\right) + M\left(\lceil \tfrac{n}{2} \rceil\right) + O(n)$$

where $M(0) + M(1) = O(1)$

$\Rightarrow M(n) \approx 2M\left(\frac{n}{2}\right) + n$

(in big-O)

Quicksort:

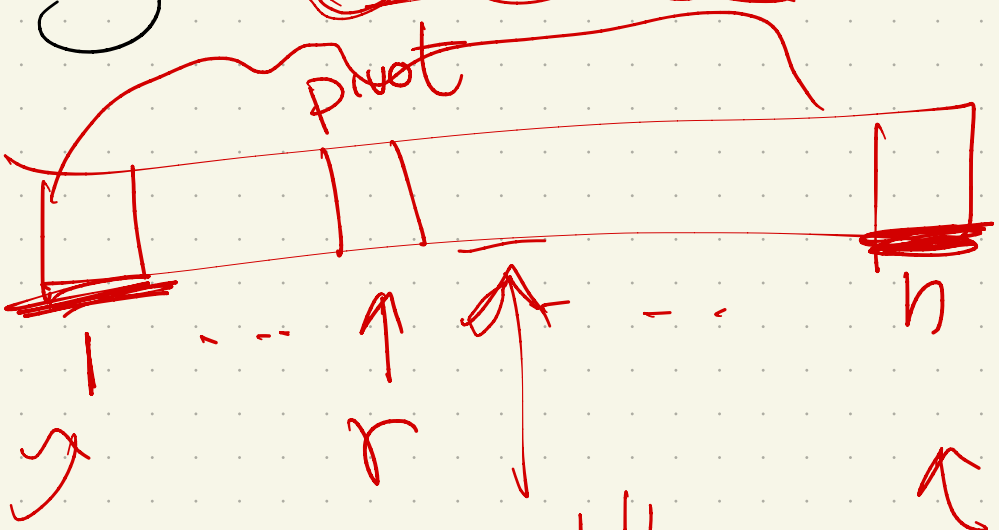
$T(\frac{n}{2})$ $T(\frac{n}{2})$

$$T(n) = \max_{1 \le r \le n} \left(T(r-1) + T(n-r) + \Theta(n) \right)$$

$T(0)$ $T(n-1)$

$T(0) = T(1) = 1$

Solving: worst case!



middle (balanced split)

unbalanced

Balanced: would have

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

(same as ~~merge~~ sort! but best case)

$\Theta(n \log n)$

Worst: $T(n) = T(0) + T(n-1) + n$

$= n + (n-1) + T(n-2) + \dots$

$\sum_{i=1}^{n} i = \Theta(n^2)$

Quicksort practicalities

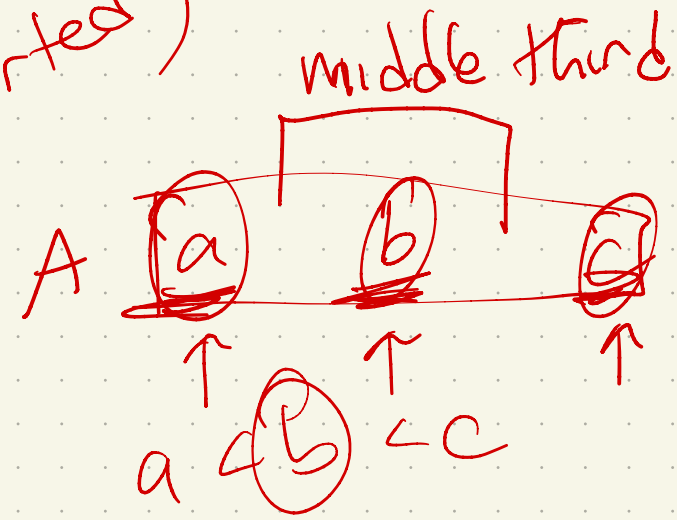
Note: "Median of three"
- Somewhat better can still be good!

Remember, while $O(n^2)$ worst case, this is the best sorting algorithm in practice.

Issues to consider: (at least outside of 3100)

- chances of a good pivot are high
- easier to implement
- space: qs can be done in place

(language) size & type of data (how sorted)



Compare a, b, c
use the middle as pivot

Recursion Trees: $T(k) = 3T(\frac{k}{2}) + k^2$

Let's start with an example.

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

n^2: nested for loops

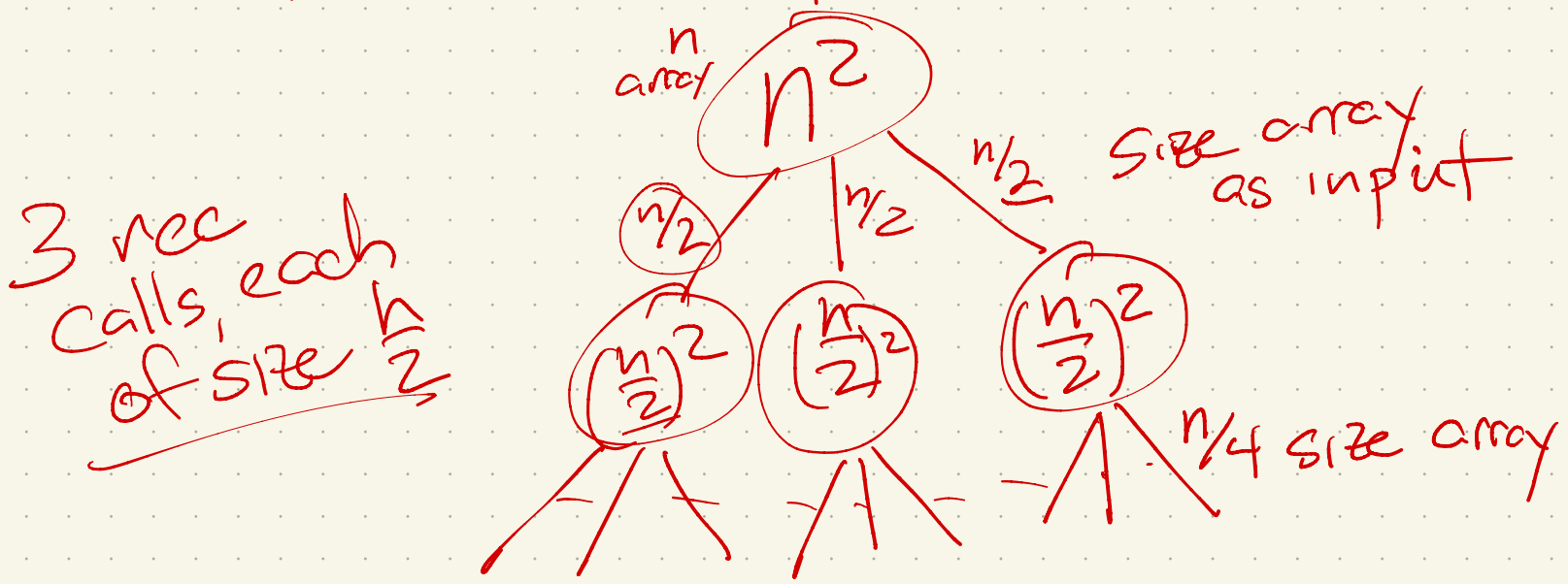
How can I "visualize" the time spent?

→ array of size n

divide into 2: $1 \ldots \frac{n}{2}$, $\frac{n}{2}+1 \ldots n$

& make 3 recursive calls

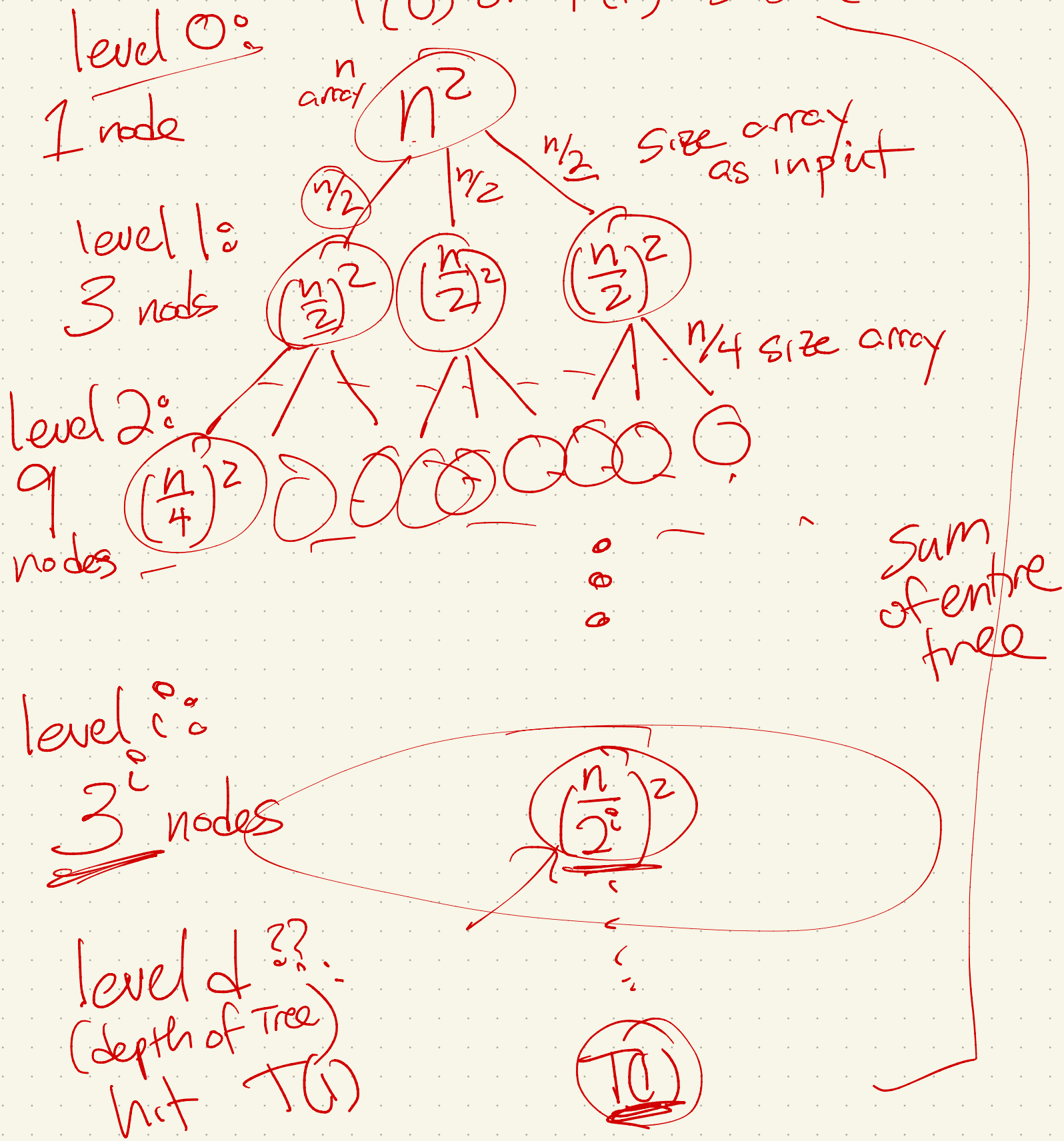
think about "top level"



(cont): Solve $T(n)$, where

$T(k) = 3T(\frac{k}{2}) + k^2$

$T(0)$ or $T(1)$ base case



Sum of all work in tree

$$= \sum_{i=0}^{d} (\#\text{ nodes on level } i)(\text{amt of work per node})$$

$$= \sum_{i=0}^{\log n} (3^i)\left(\frac{n}{2^i}\right)^2$$

Solve for d: $1 = \frac{n}{2^d}$

$2^d = n$

& take log: $\log(2^d) = \log n$

$d \log 2 = d = \log n$

(& use algebra & sums)

$$= n^2 \sum_{i=0}^{\log n} (3^i)\left(\frac{1}{2^i}\right)^2 = n^2 \sum_{i=0}^{\log n} \left(\frac{3}{4}\right)^i$$

factor out non-i terms; simplify

Note on reading:

If you don't follow the bit on ignoring floors & ceilings — don't stress!

I need you to know you <u>can</u> do this, but won't ask you to prove it.

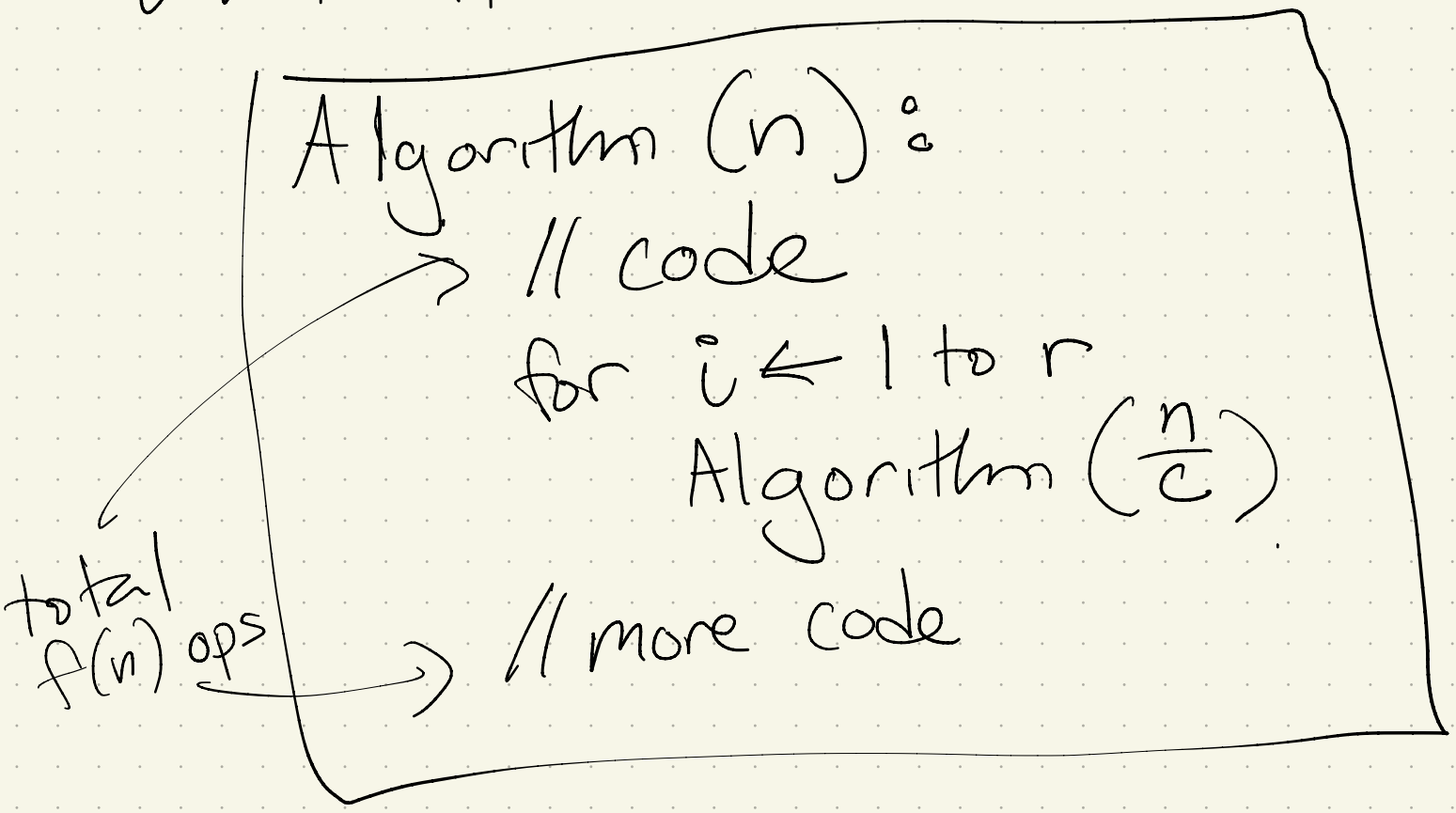
Ex: discussion on domain transformation

$$T(n) = T(\lceil \tfrac{n}{2} \rceil) + T(\lfloor \tfrac{n}{2} \rfloor) + O(n)$$

Next reading: how to generalize?

$$T(n) = rT\left(\frac{n}{c}\right) + f(n)$$

What it means:



Solving :