Algorithms, Spring '25

Recursion (cont)

Recap

- HW1: end of next week
- Reading: Going ok?
- Tentative final date:
  Thursday May 8, 10:30 am
  (Midterm still March 4 at 8 am)
- Office hours now all posted

Recursion

- If you can solve directly (usually because input is small), do it!
- Otherwise, reduce to simple (usually smaller) instances of the same problem.

Result:

Recursion Fairy

- Helps to solidify that "black box" mentality, so you don't keep unpacking the next level.

(She's also called the "induction hypothesis".)

Multiplication: How?

$$x \cdot y = \begin{cases} 0 & \text{if } x = 0 \\ \lfloor x/2 \rfloor \cdot (y+y) & \text{if } x \text{ is even} \\ \lfloor x/2 \rfloor \cdot (y+y) + y & \text{if } x \text{ is odd} \end{cases}$$

smaller!

← base case

$\lfloor \frac{x}{2} \rfloor = x/2$

Why? Proof by cases:

if x is even:

then $\exists k \in \mathbb{Z}$ s.t. $x = 2k$

$x \cdot y = (2k) \cdot y = k(2y)$

$= \lfloor \frac{x}{2} \rfloor (y+y)$

if x is odd:

then $\exists \ell \in \mathbb{Z}$ s.t. $x = 2\ell + 1$

$x = 7$

$x = 2 \cdot 3 + 1$

$\ell = \lfloor \frac{x}{2} \rfloor = 3$

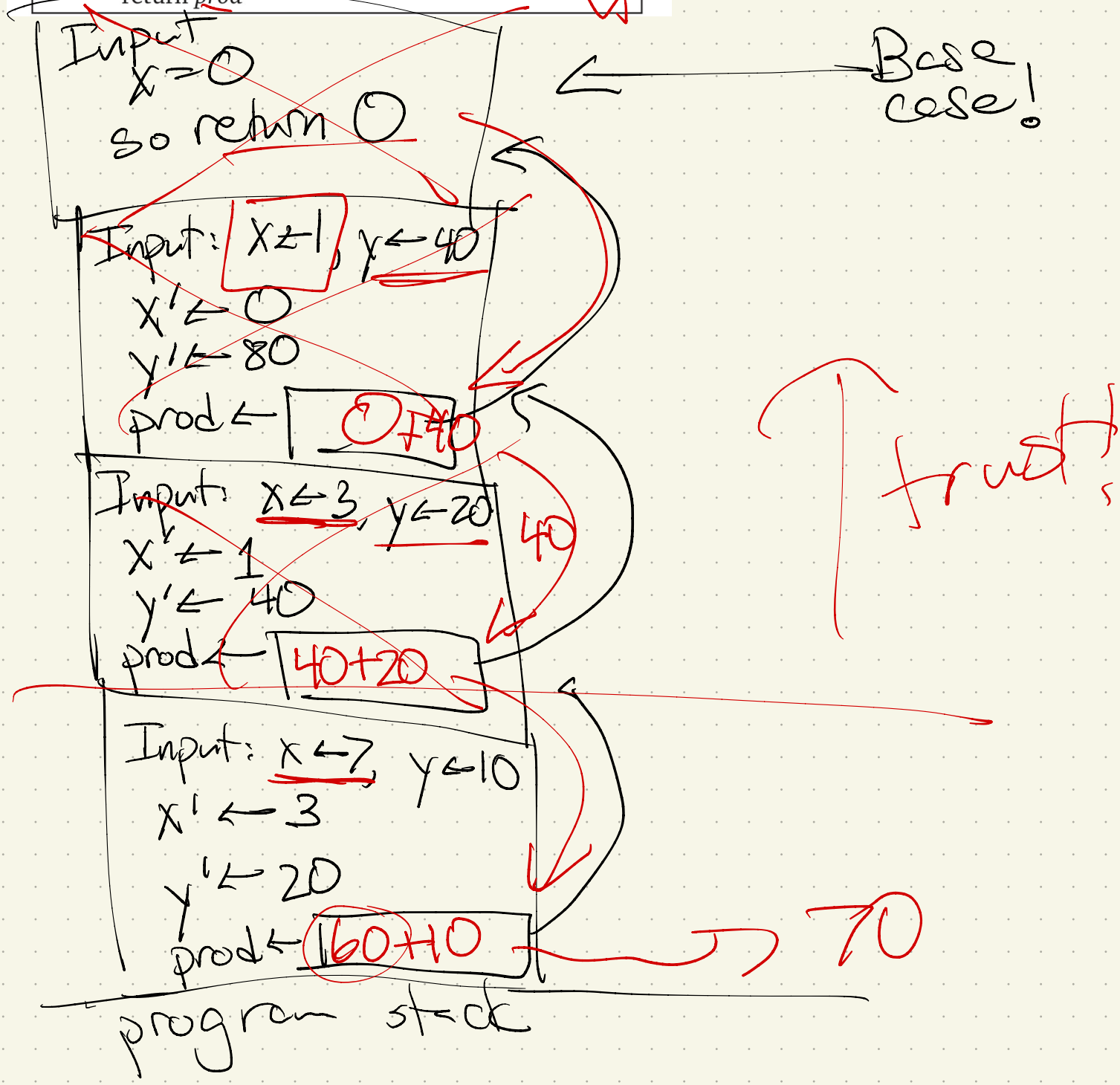
$(2\ell+1)y = 2\ell y + y$

$= \ell(2y) + y$

$= \ell(y+y) + y$

"Unpacking": Suppose we call PM(7, 10)

```
PeasantMultiply(x, y):
  if x = 0
      return 0
  else
      x' ← ⌊x/2⌋
      y' ← y + y
      prod ← PeasantMultiply(x', y')    ⟨⟨Recurse!⟩⟩
      if x is odd
          prod ← prod + y
      return prod
```



Correctness

Induction on x, for any y:

Base case: $x = 0$

$x \cdot y = 0 \cdot y = 0$ ✓

+1 return 0

IH: For values $x' < x$, the algorithm returns $x' \cdot y$.

IS: Consider $(x) \cdot y$:

If x is even:

algorithm computes $\left\lfloor \frac{x}{2} \right\rfloor \cdot (y+y)$ correctly (by IH)

and $\left\lfloor \frac{x}{2} \right\rfloor \cdot (y+y) = x \cdot y$

$x' < x$ prev proof

If odd: algorithm computes

$\left\lfloor \frac{x}{2} \right\rfloor (y+y)$ Correct by IH

and then adds $+y$

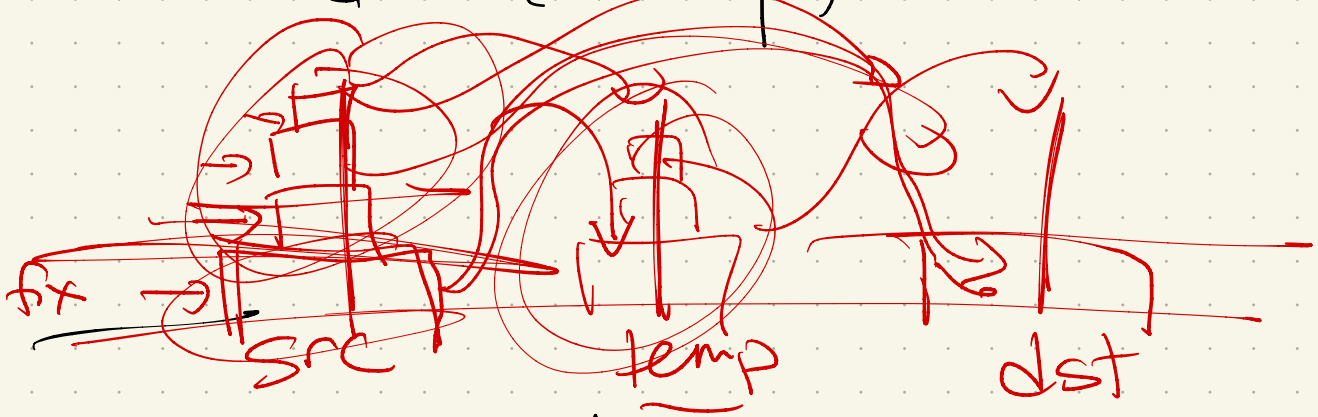
by prev formula, this is $= x \cdot y$:

Towers of Hanoi: runtime

```
HANOI(n, src, dst, tmp):
  if n > 0
      HANOI(n − 1, src, tmp, dst)     ⟨⟨Recurse!⟩⟩
      move disk n from src to dst
      HANOI(n − 1, tmp, dst, src)     ⟨⟨Recurse!⟩⟩
```

Figure 1.4. A recursive algorithm to solve the Tower of Hanoi

How ?? (no loop, + calls itself!)



Proof of correctness:

BC: if $n=1$, this will move the disk

IH: Assume I can move $n-1$ disks correctly

IS: Fix largest, use IH to move $n-1$ to tmp Now move largest, + repeat IH again

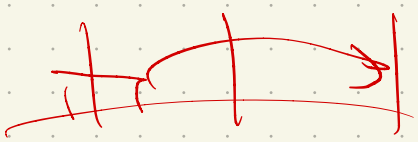
Runtime (for Hanoi):

```
HANOI(n, src, dst, tmp):
  if n > 0
    HANOI(n − 1, src, tmp, dst)    ⟨⟨Recurse!⟩⟩
    move disk n from src to dst
    HANOI(n − 1, tmp, dst, src)    ⟨⟨Recurse!⟩⟩
```

Figure 1.4. A recursive algorithm to solve the Tower of Hanoi

$H(k)$ = runtime (# moves) to move k disks.

$H(1) = 1$



$$H(n) = H(n-1) + 1 + H(n-1)$$

$$a_n = a_{n-1} + 1 + a_{n-1}$$

$$a_n = 2a_{n-1} + 1$$

$$H(n) = 2H(n-1) + 1$$

Merge Sort:

Divide & conquer recurrences + proof of correctness

```
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
  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]
```

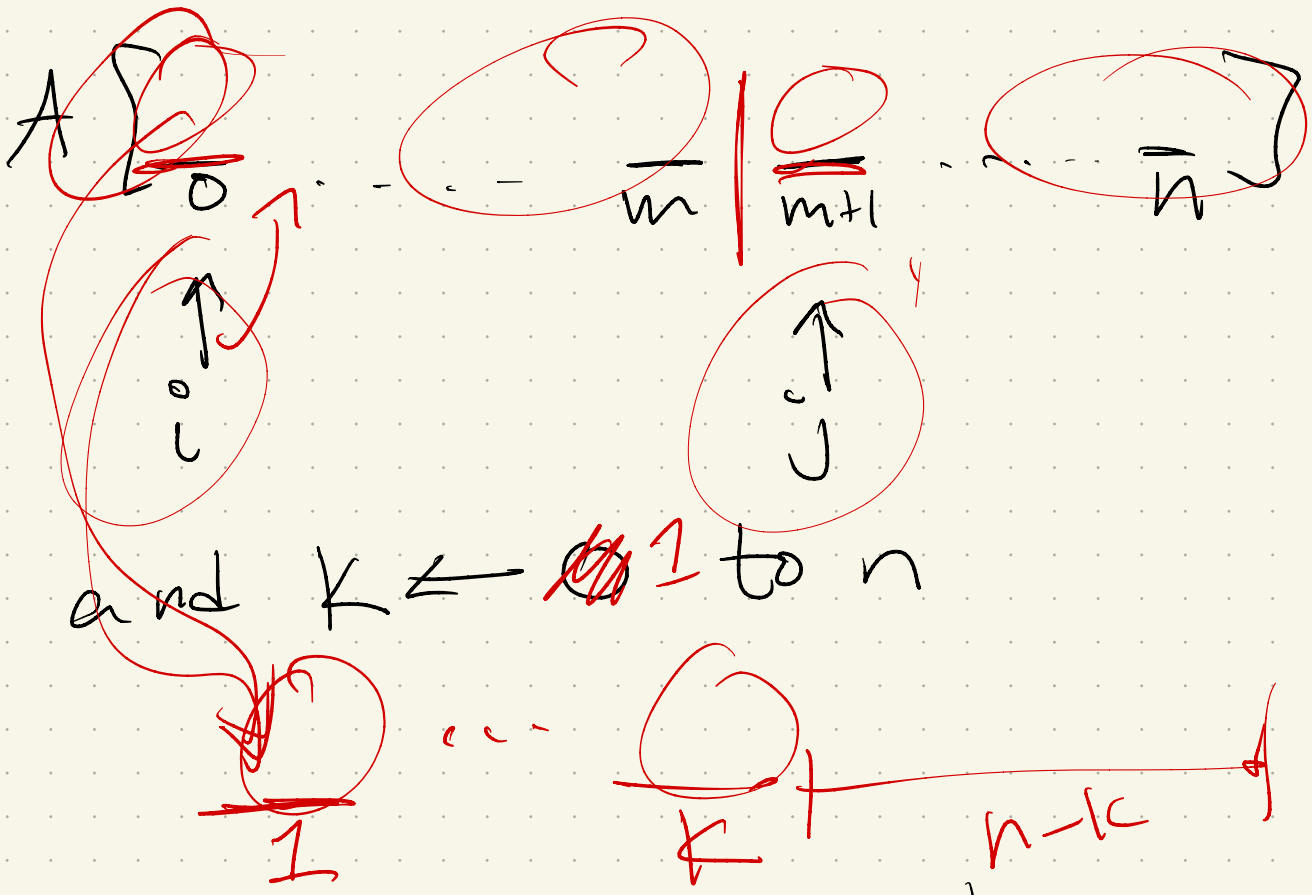
Figure 1.6. Mergesort

First: correctness, in 2 parts.

Part 1: Merge works

Setup: Given $A[1..n]$ and an index m with $1 \le m \le n$ where $A[1..m]$ & $A[m+1..n]$ are sorted, MERGE correctly sorts $A[1..n]$ by end.

How?



So: at iteration k, show we correctly copy k^{th} sorted element.

Backwards induction:
Consider what is left to sort, ie $n-k$.

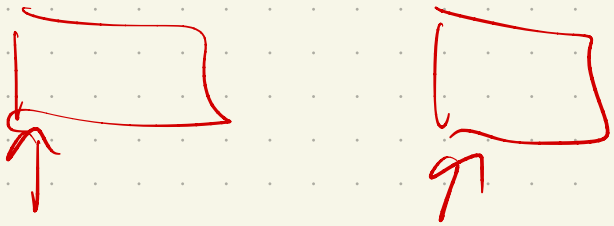
Spps $k=n$:

✓

IH:
Now, let $k < n$, & suppose works for any value greater than k:

IS: 4 cases:

```
MERGE(A[1..n], m):
  i ← 1; j ← m+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]
```



2 cases

one small list could be empty
(i = m or j = n)

Mergesort: runtime

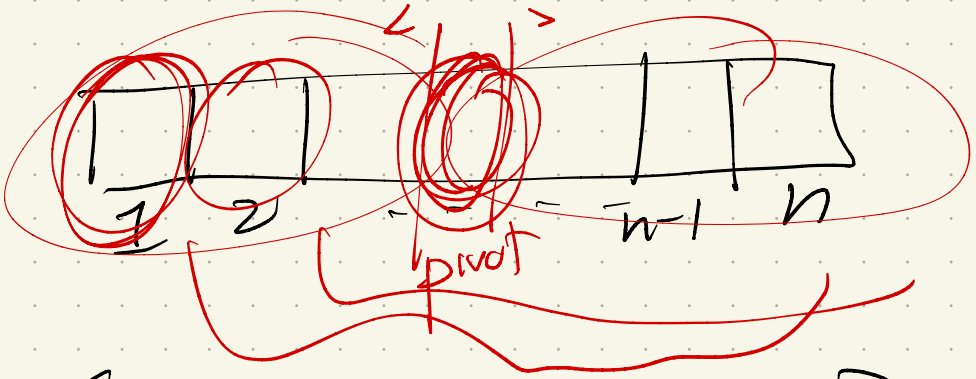
$M(k)$ = time to run mergesort on a list of size k

$$M(n) = 3 + M(\lfloor \tfrac{n}{2} \rfloor) + M(\lceil \tfrac{n}{2} \rceil) + (\text{merge time})$$

$$= 3 + 2M(\tfrac{n}{2}) + O(n)$$

$$= 2M(\tfrac{n}{2}) + O(n)$$

Quicksort:



$$T(n) \leq \max_{1 \leq r \leq n} \{ n-1 + T(r) + T(n-r) \}$$

Solving: worst case!

↳ pick largest or smallest every time

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

$$= (n-1) + (n-2) + T(n-3)$$

$$= (n+1) + (n-2) + (n-3) + T(n-4)$$

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

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 theory)

- have to be really unlucky to hit worst case each time.
- space
  - ↳ quick sort is "in place"

Recursion Trees:

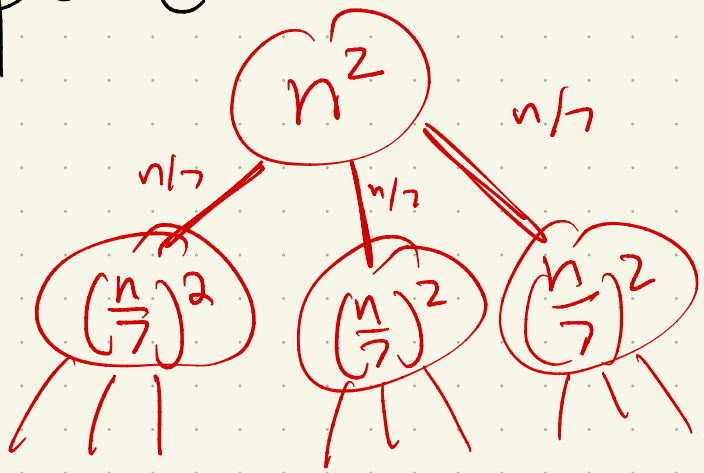
Let's start with an example.

Suppose we have a function which:

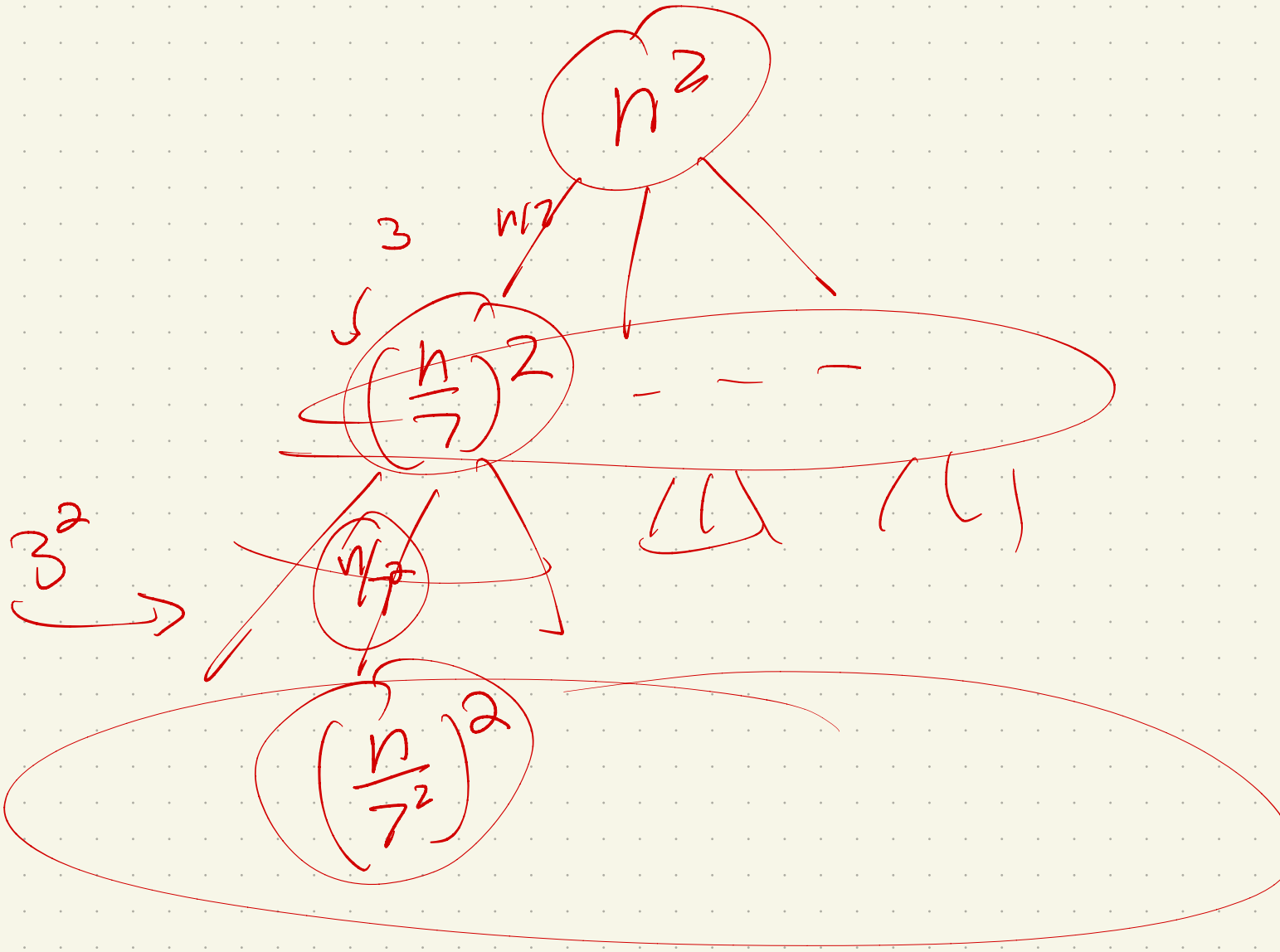
- takes input of size n
- Makes 3 recursive calls to input of size $n/7$ each
- And has a double for loop inside

$$T(n) = 3T\left(\frac{n}{7}\right) + n^2$$

How can I "visualize" the time spent?



Recursion trees (cont)



Next part: how to generalize?

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

What it means:

```
Algorithm(n):
    // code
    for i ← 1 to r
        Algorithm(n/c)
    // more code
```

Solving:

Master Theorem:

Combining the three cases above gives us the following "master theorem".

Theorem 1 *The recurrence*

$$\begin{aligned} T(n) &= aT(n/b) + cn^k \\ T(1) &= c, \end{aligned}$$

where a, b, c, and k are all constants, solves to:

$$\begin{aligned} T(n) &\in \Theta(n^k) \text{ if } a < b^k \\ T(n) &\in \Theta(n^k \log n) \text{ if } a = b^k \\ T(n) &\in \Theta(n^{\log_b a}) \text{ if } a > b^k \end{aligned}$$

THEOREM 2 **MASTER THEOREM** Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + cn^d$$

whenever $n = b^k$, where k is a positive integer, $a \geq 1$, b is an integer greater than 1, and c and d are real numbers with c positive and d nonnegative. Then

$$f(n) \text{ is } \begin{cases} O(n^d) & \text{if } a < b^d, \\ O(n^d \log n) & \text{if } a = b^d, \\ O(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

Proof: Draw the recusion tree!

Aside:

When can't I use Master theorem?

Other examples

Medians: find "middle" element.

Two were covered:

```
QUICKSELECT(A[1..n], k):
  if n = 1
      return A[1]
  else
      Choose a pivot element A[p]
      r ← PARTITION(A[1..n], p)
      if k < r
          return QUICKSELECT(A[1..r−1], k)
      else if k > r
          return QUICKSELECT(A[r+1..n], k−r)
      else
          return A[r]
```

Figure 1.12. Quickselect, or one-armed quicksort

Q: How do we know which side has the k^{th} element?