Algorithms — Spring '25

Recurrences

Recap

- HW over recursion — due Friday
- Reading: still posted through this week
- If you had technical HW issues: talk to me today!
- Office hours: need to shorten today, so 1-2:30

Recursion Trees:

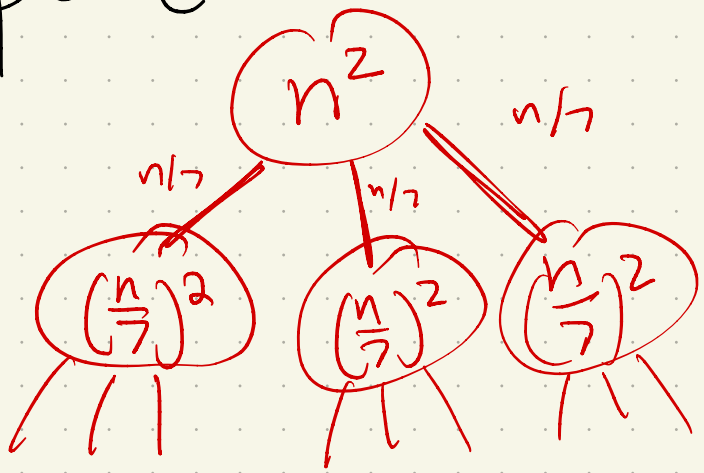
Let's start with an example.

Suppose we have a function which:

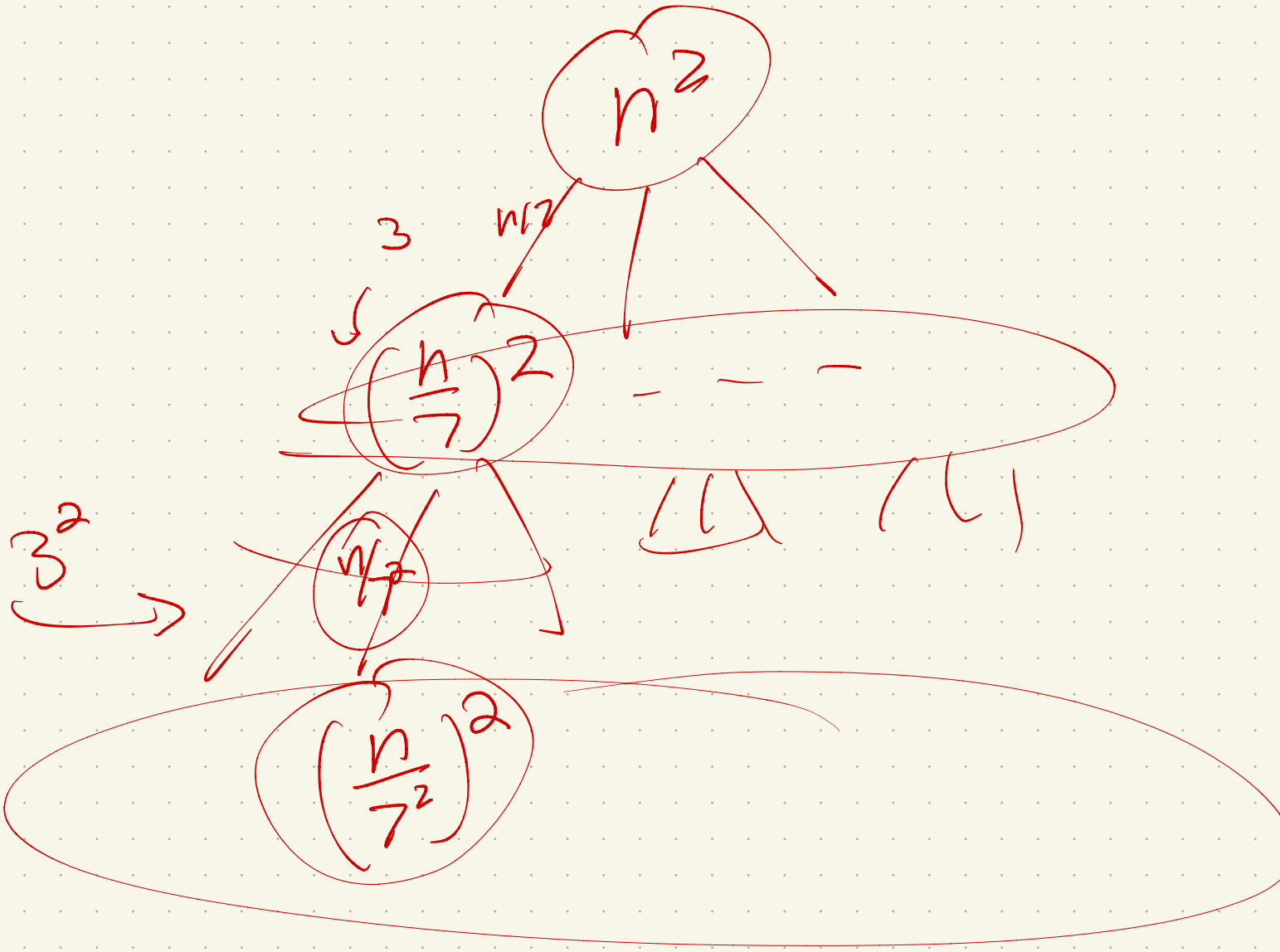
- takes input of size n
- Makes 3 recursive calls to input of size $n/7$ each
- And has a double for loop inside

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

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:

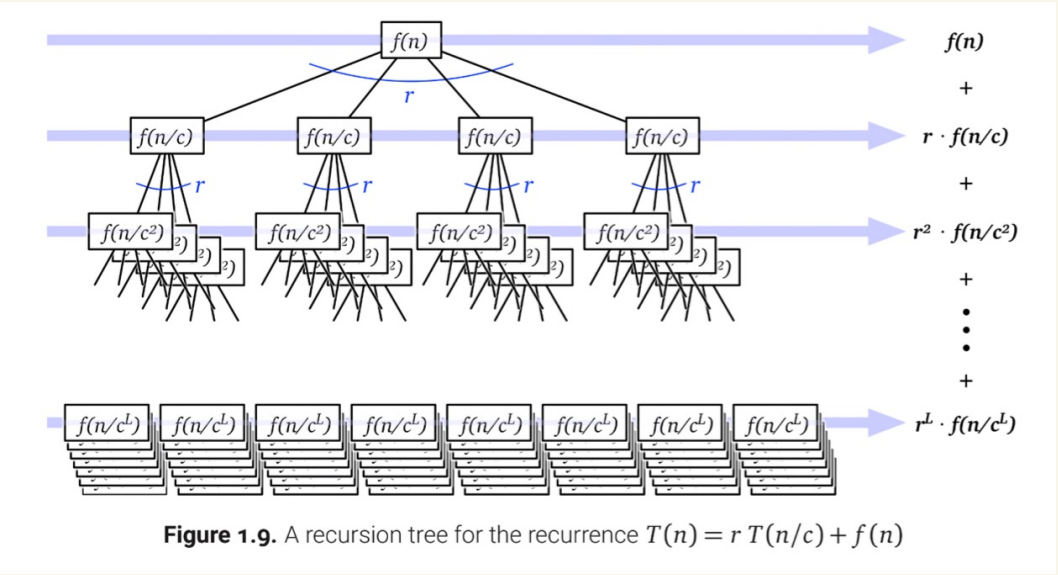


Figure 1.9. A recursion tree for the recurrence $T(n) = r\,T(n/c) + f(n)$

Geometric series:

$$\sum_{k=0}^{n} a \cdot r^k$$

All depends on r:

if $r < 1$:

$$\sum_{k=0}^{n} a \cdot r^k < \sum_{k=0}^{\infty} a \cdot r^k = \frac{a}{1-r}$$

if $r > 1$: $= \dfrac{a(r^n - 1)}{r - 1}$

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!

(& use geom series)

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?

Runtime

Still depends on pivot!

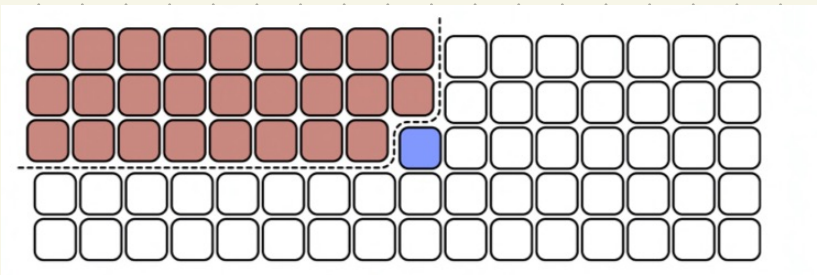
"Faster" version:

use for loop (still $O(1)$ time)

```
MomSelect(A[1..n], k):
  if n ≤ 25  ⟨⟨or whatever⟩⟩
      use brute force
  else
      m ← ⌈n/5⌉
      for i ← 1 to m
          M[i] ← MedianOfFive(A[5i−4..5i])   ⟨⟨Brute force!⟩⟩
      mom ← MomSelect(M[1..m], ⌊m/2⌋)        ⟨⟨Recursion!⟩⟩

      r ← Partition(A[1..n], mom)

      if k < r
          return MomSelect(A[1..r−1], k)      ⟨⟨Recursion!⟩⟩
      else if k > r
          return MomSelect(A[r+1..n], k−r)    ⟨⟨Recursion!⟩⟩
      else
          return mom
```



The recurrence:

```
MomSelect(A[1..n], k):
  if n ≤ 25   ⟨⟨or whatever⟩⟩
    use brute force
  else
    m ← ⌈n/5⌉
    for i ← 1 to m
      M[i] ← MedianOfFive(A[5i−4..5i])   ⟨⟨Brute force!⟩⟩
    mom ← MomSelect(M[1..m], ⌊m/2⌋)      ⟨⟨Recursion!⟩⟩
    r ← Partition(A[1..n], mom)
    if k < r
      return MomSelect(A[1..r−1], k)     ⟨⟨Recursion!⟩⟩
    else if k > r
      return MomSelect(A[r+1..n], k−r)   ⟨⟨Recursion!⟩⟩
    else
      return mom
```

Multiplication:

Known "fact":

$$(10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc + ad) + bd$$

Example:

$$102568 \times 358691 =$$

↳ Why does this suggest recursion??

The algorithm:

```
SplitMultiply(x, y, n):
  if n = 1
      return x · y
  else
      m ← ⌈n/2⌉
      a ← ⌊x/10^m⌋;  b ← x mod 10^m        ⟨⟨x = 10^m a + b⟩⟩
      c ← ⌊y/10^m⌋;  d ← y mod 10^m        ⟨⟨y = 10^m c + d⟩⟩
      e ← SplitMultiply(a, c, m)
      f ← SplitMultiply(b, d, m)
      g ← SplitMultiply(b, c, m)
      h ← SplitMultiply(a, d, m)
      return 10^(2m) e + 10^m (g + h) + f
```

Runtime:

A better trick:

$$ac + bd - (a-b)(c-d) = bc + ad$$

```
FastMultiply(x, y, n):
  if n = 1
      return x · y
  else
      m ← ⌈n/2⌉
      a ← ⌊x/10^m⌋;  b ← x mod 10^m        ⟨⟨x = 10^m a + b⟩⟩
      c ← ⌊y/10^m⌋;  d ← y mod 10^m        ⟨⟨y = 10^m c + d⟩⟩
      e ← FastMultiply(a, c, m)
      f ← FastMultiply(b, d, m)
      g ← FastMultiply(a − b, c − d, m)
      return 10^(2m) e + 10^m (e + f − g) + f
```

Runtime:

Exponentiation:

Still open!

(Amazing, right??)

The algorithms do very well:

- to compute a^n, need $O(\log n)$ multiplications

However, doesn't achieve lowest possible for every value - it's just with a constant!