Complexity & Algorithms, Spring 2026

Rest of intro
Recurrences & Recursion

Recap

- How was reading?
- HW0 - due Thursday
- Office hours posted
  - Since none yesterday, will plan to be around Thurs in morning

Clarification from last time

Big-O: $f(n)$ is $O(g(n))$ if $\exists c, N \geq 0$ s.t.

$\forall n > N$, $f(n) \leq c \cdot g(n)$

Omega: $f(n)$ is $\Omega(g(n))$ if $\exists c, N \geq 0$

s.t. $\forall n > N$, $f(n) \geq c \cdot g(n)$

Theta: $f(n)$ is $\Theta(g(n))$ if $f(n)$ is $O(g(n))$

and $f(n)$ is $\Omega(g(n))$

Little-o: $f(n) = o(g(n))$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} \to \infty$

So:

Cleaner example:

Are these $\Theta(n^2)$, $\Omega(n^2)$, $o(n^2)$?

$f(n) = 17n + 11$:

$g(n) = n \log n$:

$h(n) = \frac{x^2}{4} - 100$:

$j(n) = 3x^4/1000$:

Another: Every rooted binary tree of height h has $\leq 2^{h+1} - 1$ nodes

Recall: height$(T) = \begin{cases} \\ \\ \\ \end{cases}$

Proof:

5 Pseudo code & runtime:

Discrete math examples (from Rosen textbook)

ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

procedure $max(a_1, a_2, \ldots, a_n$: integers)
$max := a_1$
for $i := 2$ **to** n
    if $max < a_i$ **then** $max := a_i$
return max {max is the largest element}

This book:

FIBONACCIMULTIPLY$(X[0..m-1], Y[0..n-1])$:
  $hold \leftarrow 0$
  for $k \leftarrow 0$ to $n+m-1$
    for all i and j such that $i+j=k$
      $hold \leftarrow hold + X[i] \cdot Y[j]$
    $Z[k] \leftarrow hold \bmod 10$
    $hold \leftarrow \lfloor hold/10 \rfloor$
  return $Z[0..m+n-1]$

Pseudocode conventions here:

Variable assignment:

Boolean comparison:

Arrays: A[0..n-1]
- each element:

Loops:

Pseudocode format:

In a pinch, pretend you're in Python or Ruby → high level & readable.

I realize this is not a "definition" - that is the point!

It's about effective communication.

Reading today: recursion

Most of you indicated you'd seen it before. Topics here:

- Towers of Hanoi
- Merge sort
- Recap of recurrences & "Master theorem"
- Linear time selection
- Multiplication (again)
- Exponentiation

(Question: All review?)

A high level note on recursion:

Recursion really can be simpler & useful!

often depends upon the language and setup.

Counter-intuitive, but that's often due to lack of practice.

Often considered slower?

Recursion

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

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".)

Classic example

Our book

```
QUICKSORT(A[1..n]):
  if (n > 1)
      Choose a pivot element A[p]
      r ← PARTITION(A, p)
      QUICKSORT(A[1..r−1])      ⟨⟨Recurse!⟩⟩
      QUICKSORT(A[r+1..n])      ⟨⟨Recurse!⟩⟩
```

```
PARTITION(A[1..n], p):
  swap A[p] ↔ A[n]
  ℓ ← 0                         ⟨⟨#items < pivot⟩⟩
  for i ← 1 to n−1
      if A[i] < A[n]
          ℓ ← ℓ + 1
          swap A[ℓ] ↔ A[i]
  swap A[n] ↔ A[ℓ + 1]
  return ℓ + 1
```

Algorithm 1 Quicksort

```
1: procedure QUICKSORT(A, p, r)
2:     if p < r then
3:         q = PARTITION(A, p, r)
4:         QUICKSORT(A, p, q − 1)
5:         QUICKSORT(A, q + 1, r)
6:     end if
7: end procedure
8: procedure PARTITION(A, p, r)
9:     x = A[r]
10:    i = p − 1
11:    for j = p to r − 1 do
12:        if A[j] < x then
13:            i = i + 1
14:            exchange A[i] with A[j]
15:        end if
16:        exchange A[i] with A[r]
17:    end for
18: end procedure
```

QuickSort Pseudocode Example

Another version

Aside: Why 2 proofs?

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

$T(n) =$

How can I "visualize" the time spent?

Recursion tree: Sum up all operations

Recall: geometric series

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

So:

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
```

Then, turn into summation

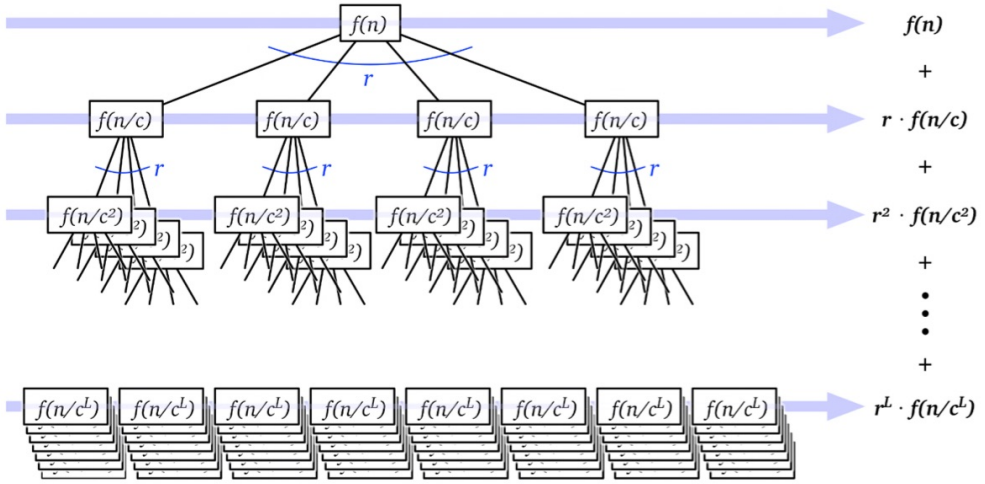


Figure 1.9. A recursion tree for the recurrence $T(n) = r\,T(n/c) + f(n)$

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:

Aside: When can't I use Master theorem?

Answer: When it's not a geometric series!

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

Can we still solve? How?

Another: $T(n) = \sqrt{n}\, T(\sqrt{n}) + O(n)$

Why?

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

Why?

Takeaway:

- Many ways to tackle recurrences
- In this class, divide & conquer (& perhaps linear inhomogeneous) will be most common
- Many other techniques exist
  - ↳ see supplemental reading if curious

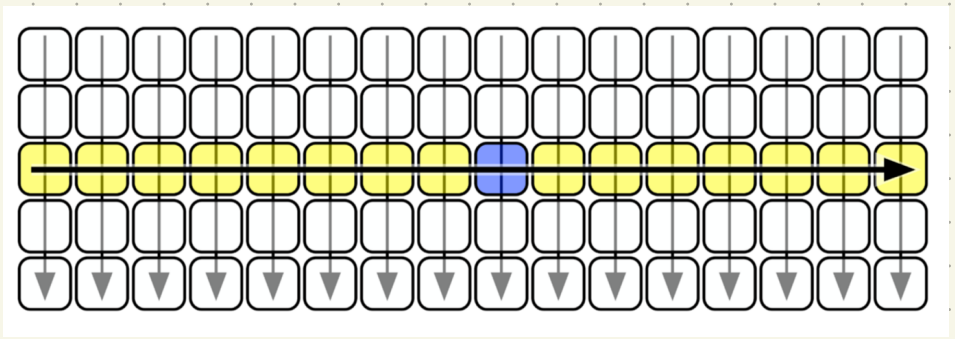
A note on MoM

Goal is to eliminate a constant fraction of the options.

How? (Can't sort!)

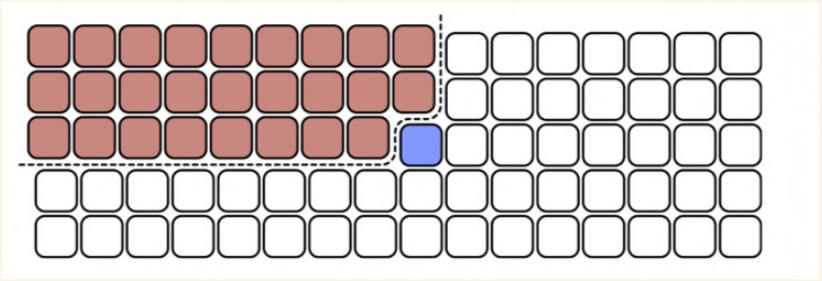
```
MomSelect(A[1..n], k):
  if n ≤ 25   ⟨⟨or whatever⟩⟩
      use brute force
  else
      m ← ⌈n/5⌉
      for i ← 1 to m
          M[i] ← MedianOfFive(A[5i−4..5i])   ⟨⟨Brute force!⟩⟩
      mom ← MomSelect(M[1..m], ⌊m/2⌋)        ⟨⟨Recursion!⟩⟩
      r ← Partition(A[1..n], mom)
      if k < r
      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

Array $A[1..n]$



First example of non-Master theorem!

Can always guarantee at least $\frac{3n}{10}$ are eliminated.



So:

$M(n) \leq$

Then solving:

Next reading: Backtracking
(will feel similar to classic AI)

Really, more recursion!
Also, helps to set up Dynamic Programming.