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

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 $\Omega(g(n))$ $:\equiv$

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

so: $f(n) = o(g(n)) \Rightarrow f(n) = \Omega(g(n))$

Cleaner example:

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

$f(n) = 17n + 11$: $O(n^2)$ $\lim_{n\to\infty} \frac{17n+11}{n^2} \to 0$

$g(n) = n \log n$: $O(n^2)$

$n \log n < n \cdot n$ $\log n < n$

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

$\hookrightarrow \Theta(n^2)$

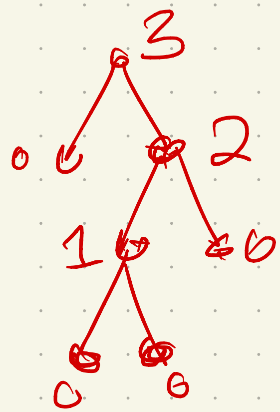
$j(n) = 3x^4/1000$: $\Omega(n^2)$

Induction!

2 or less children

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

Recall: $\text{height}(T) = \begin{cases} 0 & \text{if no children} \\ \max\limits_{\text{children } x} \big(h(x)\big) + 1 \end{cases}$



$2^{3+1}-1$

≤ 15 nodes

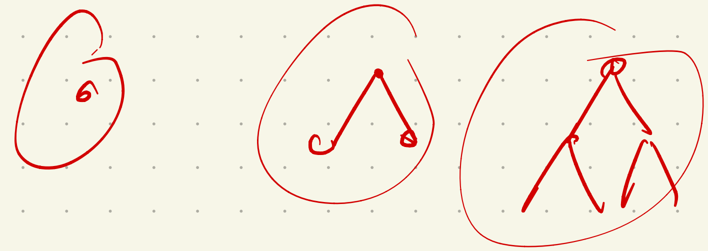
Proof: Induction on height h of tree.

Base case: $h = 0$

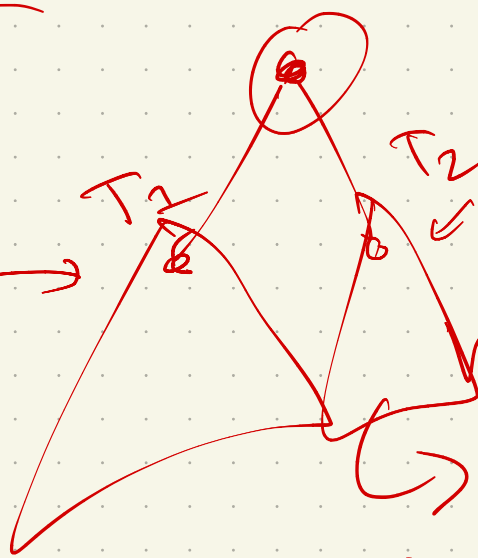
1 node $\le 2^{0+1}-1 = 1$

IH: tree with height $k < h$ has $\leq 2^{k+1} - 1$ nodes

IS: Consider height h tree T:



T



Use IH: $\#$ nodes in $T_1 \leq 2^{(h-1)+1} - 1$

$\#$ nodes in $T_2 \leq 2^{i+1} - 1 \leq 2^{(h-1)+1} - 1$

$$\Rightarrow \# \text{nodes in } T \leq 1 + \overbrace{(2^h - 1)}^{T_1} + \overbrace{(2^h - 1)}^{T_2} = 2^1 \cdot 2^h - 1 = 2^{h+1} - 1 \quad \square$$

5) Pseudo code + runtime:

Discrete math examples (from Rosen textbook)

ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

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

x == 2
x = 2
boolean

Pascal-like

This book:

var. assignment

function calls

$\textsc{FibonacciMultiply}(X[0..m-1], Y[0..n-1])$:
  $hold \leftarrow 0$
  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 $Z[0..m+n-1]$

boolean

Pseudocode conventions here:

Variable assignment: $\leftarrow$

Boolean comparison: $x = y$ or $x == y$

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

Loops: for $i \leftarrow 1$ to n

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) ↳ FFT
- 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: memory!

Not really fair

↳ functional languages

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.

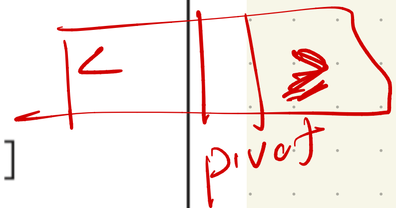
(She's also called the "induction hypothesis".)

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
```



Another version

Algorithm 1 Quicksort

```
procedure Quicksort(A, p, r)
    if p < r then
        q = Partition(A, p, r)
        Quicksort(A, p, q − 1)
        Quicksort(A, q + 1, r)
    end if
end procedure
procedure Partition(A, p, r)
    x = A[r]
    i = p − 1
    for j = p to r − 1 do
        if A[j] < x then
            i = i + 1
            exchange A[i] with A[j]
        end if
        exchange A[i] with A[r]
    end for
end procedure
```

QuickSort Pseudocode Example

Aside: Why 2 proofs?

2 functions!

in Mergesort

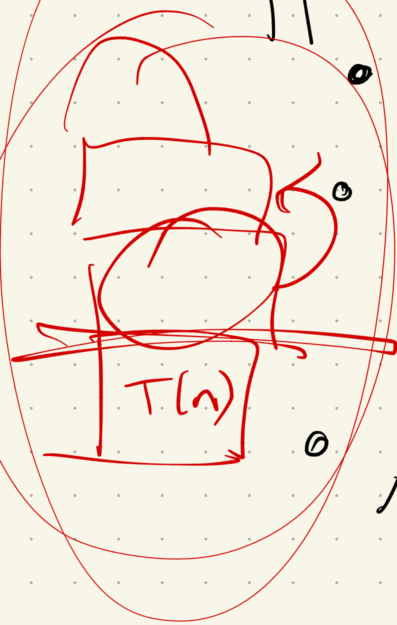
Recursion Trees:

Let's start with an example.

Suppose we have a function which:

- takes input of size n $A[1..n]$
- Makes 3 recursive calls to input of size $n/7$ each
- And has a double for loop inside

  for $i \leftarrow 1$ to n
  for $j \leftarrow 1$ to i

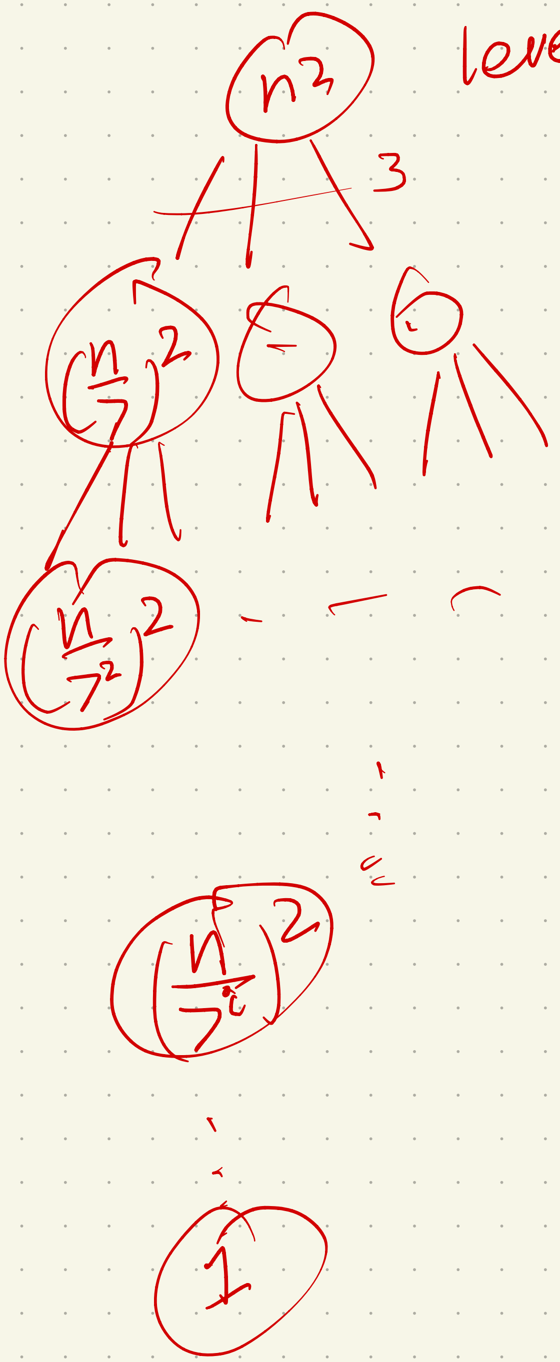


$$T(n) = \text{"top level"} + \text{rec calls} = 3T\left(\frac{n}{7}\right) + n^2$$

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

How can I "visualize" the time spent?

Recursion tree: Sum up all operations



level 0

$$\left(\frac{1}{7^i}\right)^2$$

level 1 $= \frac{1}{7^{2i}} = \frac{1}{(7^2)^i}$

level 2

level i ↙
3^i nodes

depth d:

$$\frac{n}{7^d} = 1 \Rightarrow n = 7^d$$
$$d = \log_7 n$$

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

$$= \sum_{i=0}^{\log_7 n} 3^i \left(\frac{n}{7^i}\right)^2$$

$$= \sum_{i=0}^{\log_7 n} 3^i \cdot \frac{1}{49^i} \cdot n^2$$

$$= n^2 \sum_{i=0}^{\log_7 n} \left(\frac{3}{49}\right)^i$$

$$\leq n^2 \sum_{i=0}^{\infty} \left(\frac{3}{49}\right)^i$$

$$= n^2 \cdot \frac{1}{1-\frac{3}{49}}$$

$$= \Theta(n^2)$$

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1-r}$$

if $r < 1$

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: if summation looks like series, can solve

Next part: how to generalize?

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

\# of rec calls

What it means:

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

Then, turn into summation

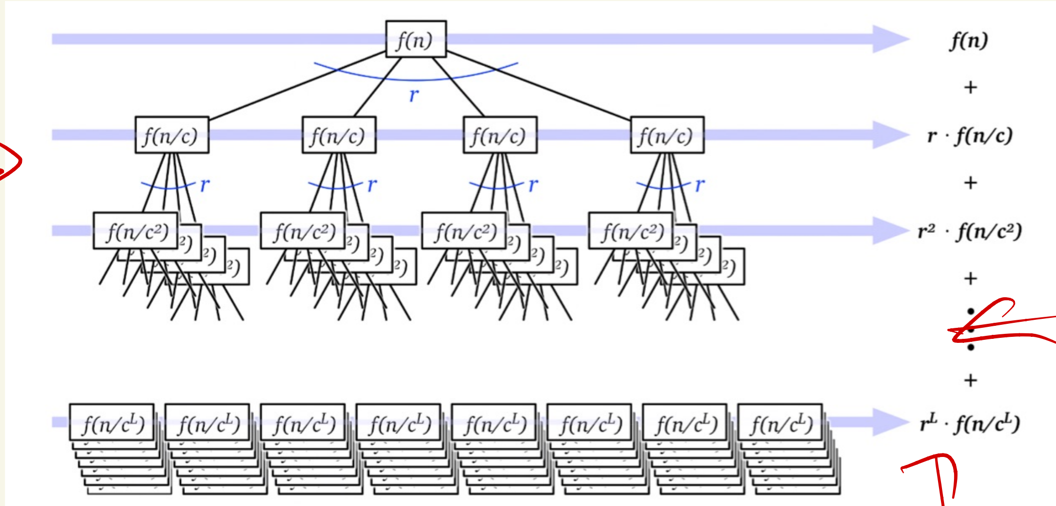


Figure 1.9. A recursion tree for the recurrence $T(n) = r\,T(n/c) + f(n)$

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

level i: r^i nodes, each doing $f\left(\frac{n}{c^i}\right)$ operations

depth $= L$

$$\frac{n}{c^L} = 1 \Rightarrow L = \log_c n$$

$$T(n) = \sum_{i=0}^{L=\log_c n} r^i f\left(\frac{n}{c^i}\right)$$

is this a geom series?

Master Theorem:

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

Theorem 1 *The recurrence*

$$T(n) = aT(n/b) + cn^k$$
$$T(1) = c,$$

$\sqrt{n}$ $\sqrt{n}$ $n/3$ n^2

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

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

← descending geom series

← ratio = 1

asending geom series

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

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

$c = 1$

$$\sum_{i=1}^{d} c^i$$

Proof: geom series

Aside: When <u>can't</u> I use Master theorem?

Answer: When it's not a geometric series!

linear recurrences

inhomogeneous

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

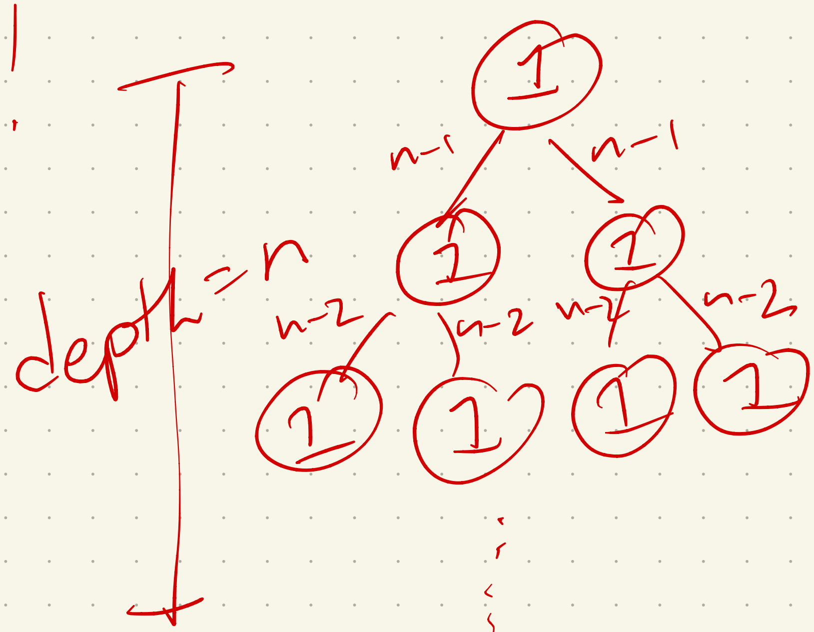
Can we still solve? How?

characteristic eqn method

exponential!

has $2^{n+1} - 1$ nodes

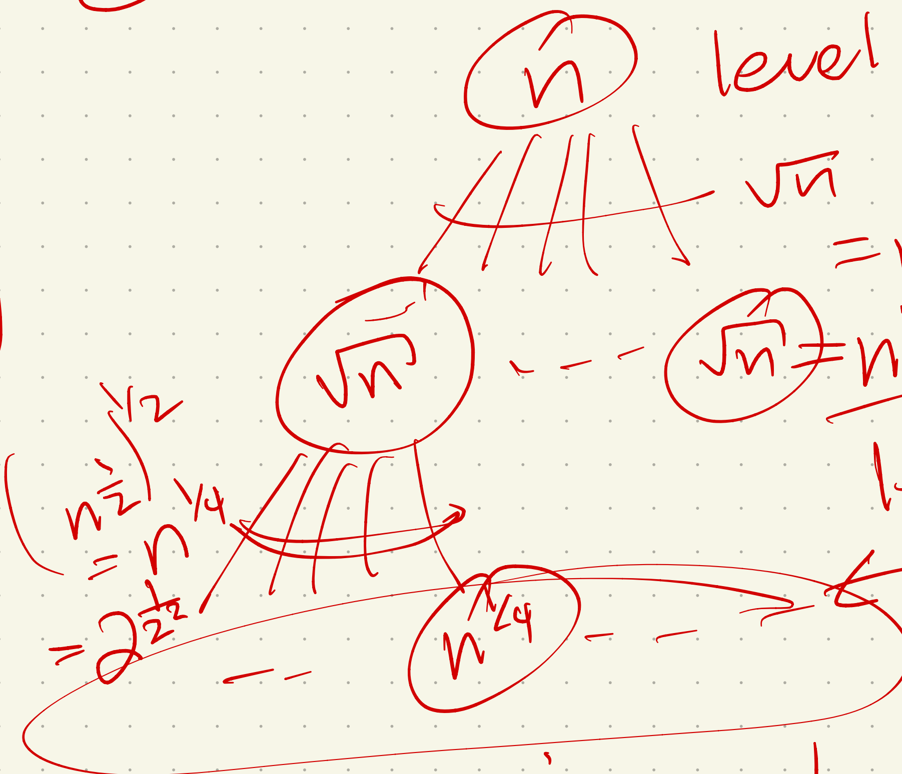
$= \Theta(2^n)$



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

Why? no r or c

Still do tree:



level 0

$\sqrt{n}$ children $= n^{1/2}$

level 2

$(n^{1/2})(n^{1/4}) = n^{3/4}$

level i

$n^{1-\frac{1}{2^i}}$ nodes

$n^{\frac{1}{2^i}}$

$$\sum_{i=0}^{\log\log n} \underbrace{n^{\frac{1}{2^i}}}_{\text{work per node}} \underbrace{\left(n^{1-\frac{1}{2^i}}\right)}_{\#\text{ nodes}}$$

$$= \sum_{i=0}^{\log\log n} n$$

$$= n \sum_{i=0}^{\log\log n} 1$$

$$= n \log\log n$$

$(\log n)^2 = \log^2 n$

$n^{\frac{1}{2^d}} = 1_{2^d}$

$1 = n$

depth: $d = \log(\log n)$

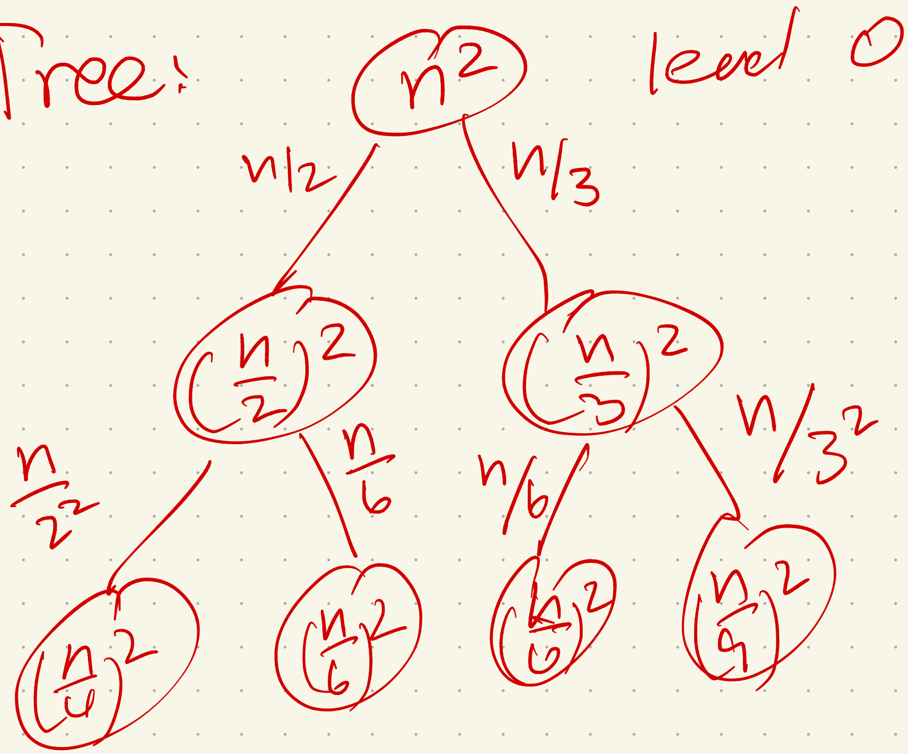
$2^d = \log n$

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

Why? 2 rec calls

↳ different sizes

Tree:



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

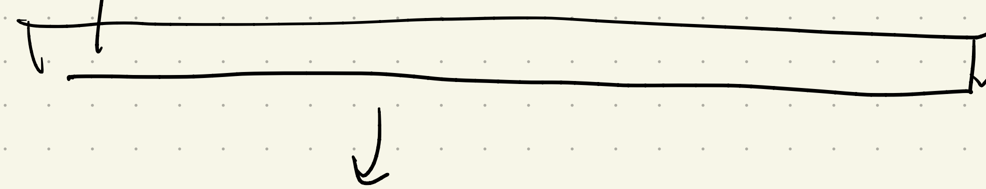
A note on MoM

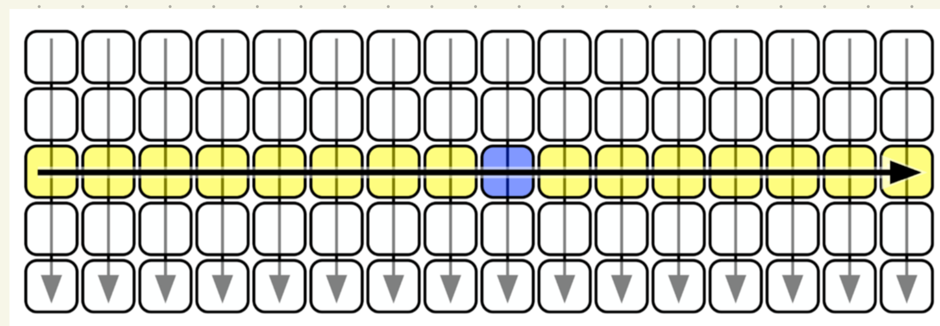
Goal is to eliminate a constant fraction of the options.

How? (Can't sort!)

MomSelect($A[1\,..\,n], k$):
  if $n \le 25$ ⟨⟨or whatever⟩⟩
    use brute force
  else
    $m \leftarrow \lceil n/5 \rceil$
    for $i \leftarrow 1$ to m
      $M[i] \leftarrow$ MedianOfFive($A[5i-4\,..\,5i]$) ⟨⟨Brute force!⟩⟩
    $mom \leftarrow$ MomSelect($M[1\,..\,m], \lfloor m/2 \rfloor$) ⟨⟨Recursion!⟩⟩
    $r \leftarrow$ 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

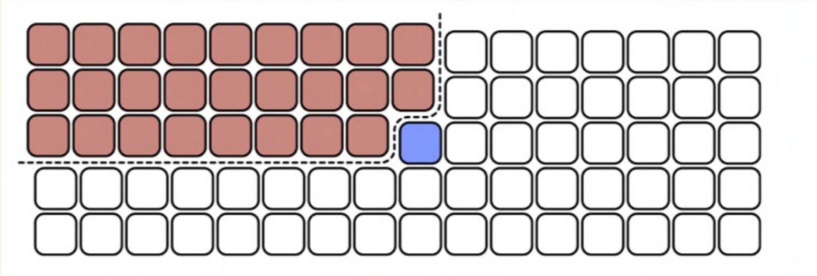
Array $A[1..n]$





First example of non-Master theorem!

Can alway 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.