Algorithms & Complexity, Spring 2026

Recursion
Backtracking

Recap

- HW0 due tonight
  - ↳ office hours after class
- Next readings: posted, & a bit shorter
  - ↳ Dynamic programming
- Week after - will switch to new book
  - ↳ Greedy approximations

- HW1: Recursion
  - ↳ Posted tomorrow

Last Time: Runtimes for recursive algorithms

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

r = # of rec calls

What it means:

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

Then, turn into summation

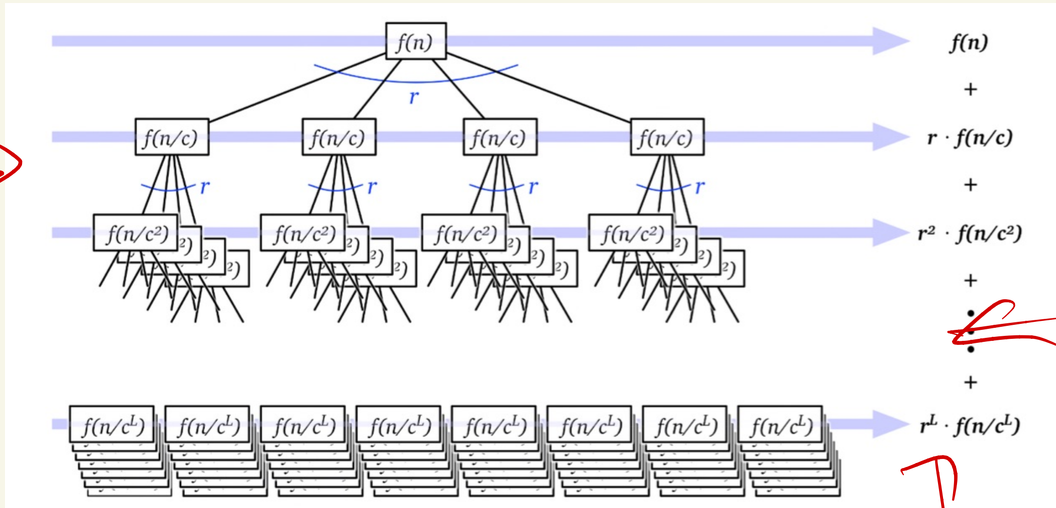


Figure 1.9. A recursion tree for the recurrence $T(n) = r\,T(n/c) + f(n)$

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

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

depth $= L$

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

$$\Downarrow$$

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

is this a geom series?

Master Theorem: Classify by looking at recurrence more quickly

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

Theorem 1 *The recurrence*

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

$\sqrt{n}$ $n\sqrt{3}$ 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

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

$c = 1$

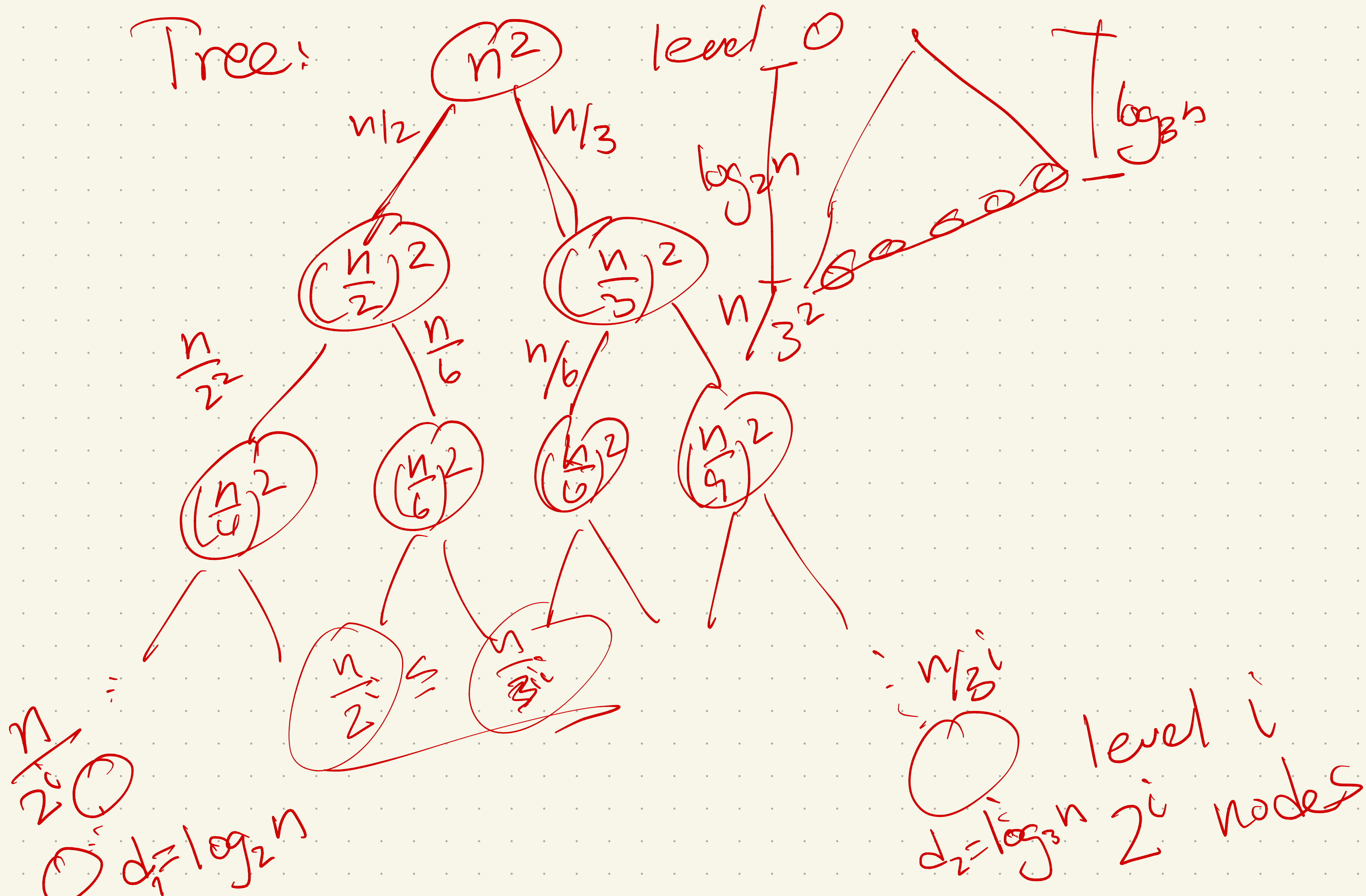
$$\sum_{i=1}^{d} c^i$$

Proof: geom series

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

Why? 2 rec calls
↳ different sizes

Tree:



$$\sum_{i=0}^{\log_3 n} 2^i \underbrace{\left(\frac{n}{3^i}\right)^2}_{\text{work per node}} \le T(n) \le \sum_{i=0}^{\log_2 n} 2^i \underbrace{\left(\frac{n}{2^i}\right)^2}_{\text{work per node}}$$

$$T(n) \le \sum_{i=0}^{\log_2 n} 2^i \left(\frac{n}{2^i}\right)^2 = \sum_{i=0}^{\log_2 n} n^2 \cdot 2^i \left(\frac{1}{2^{2i}}\right)$$

$$= n^2 \sum_{i=0}^{\log_2 n} 2^i \left(\frac{1}{4^i}\right) = n^2 \underbrace{\sum_{i=0}^{\log_2 n} \left(\frac{1}{2}\right)^i}_{\text{geom series!}}$$

ratio $r = \frac{1}{2}$

$$\le n^2 \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i = n^2 \left(\frac{1}{1-\frac{1}{2}}\right)$$

$$= O(n^2)$$

$$T(n) \geq \sum_{i=0}^{\log_3 n} 2^i \left(\frac{n}{3^i}\right)^2 = \sum_{i=0}^{\log_3 n} 2^i \cdot n^2 \cdot \left(\frac{1}{9}\right)^i$$

$$= n^2 \sum_{i=0}^{\log_3 n} \left(\frac{2}{9}\right)^i$$

~~$r < 1$ geom~~

constant > 0

$$\geq n^2 \sum_{i=0}^{1} \left(\frac{2}{9}\right)^i = c \cdot n^2$$

$T(n)$ is $\Omega(n^2)$

So: $T(n)$ is $\Theta(n^2)$

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, but most will fall into categories like you read

A note on MoM

Median of median — find k^{th} element

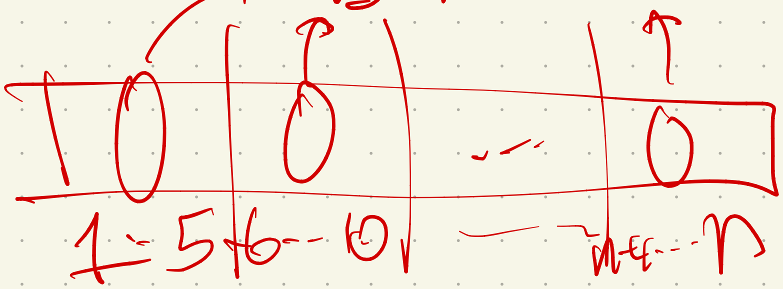
Goal is to eliminate a constant fraction of the options.

How? (Can't sort!)

Idea: Split into tiny pieces & hope median is good enough.

$n/5$ medians

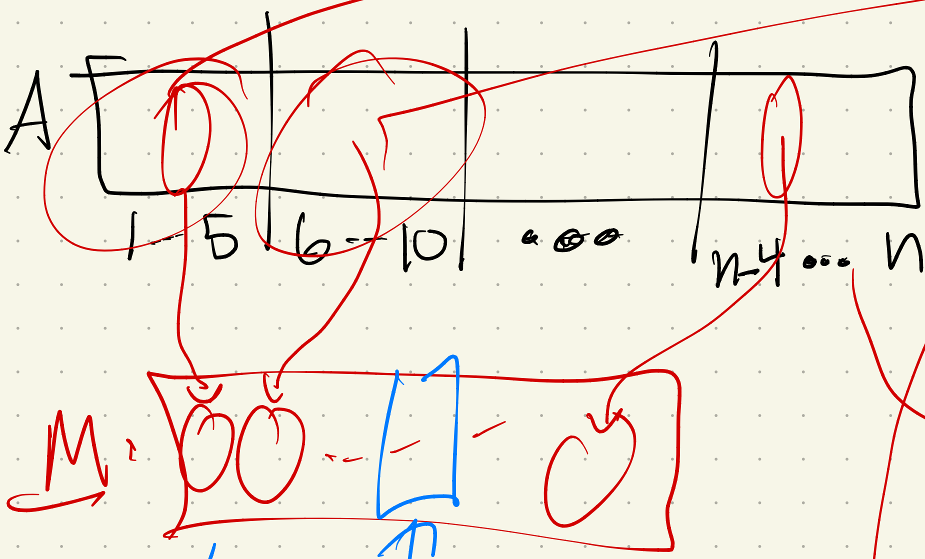
Here:

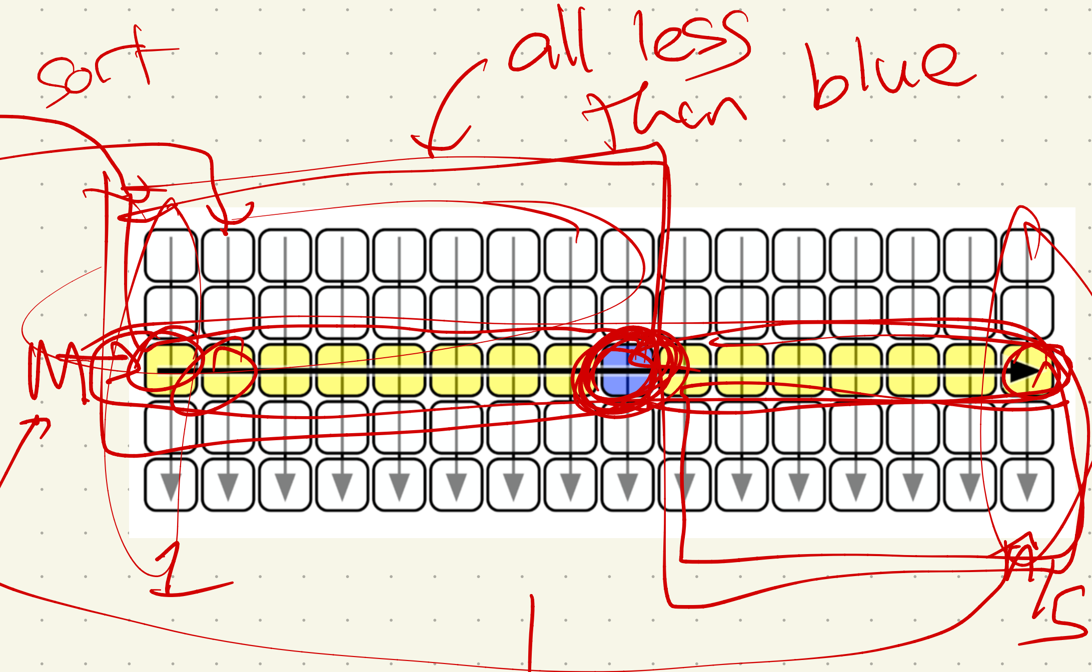


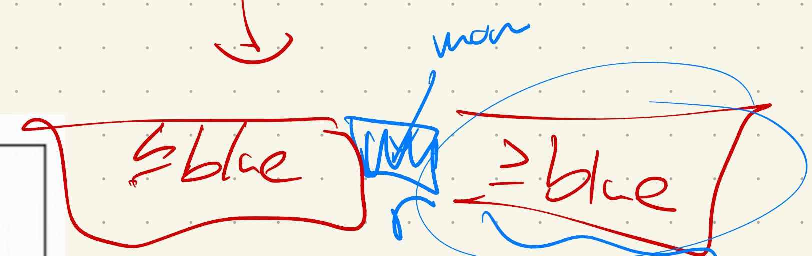
split into "small" pieces

small: find 3^{rd} of 5

Turning into code







$O(n)$ r-1 n-r-1

```
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
```

$O(n)$

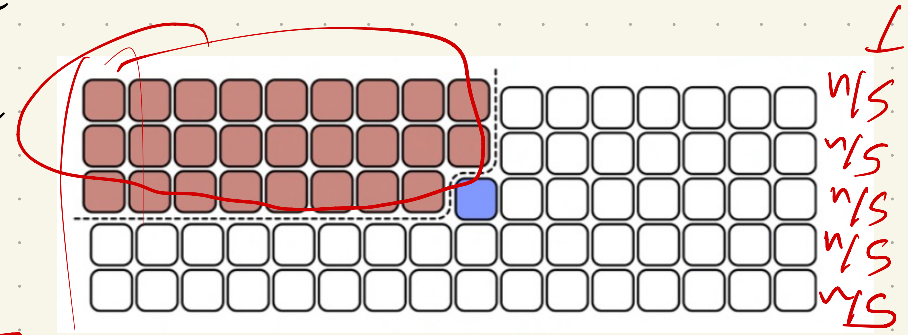
$M(\frac{n}{5})$ size

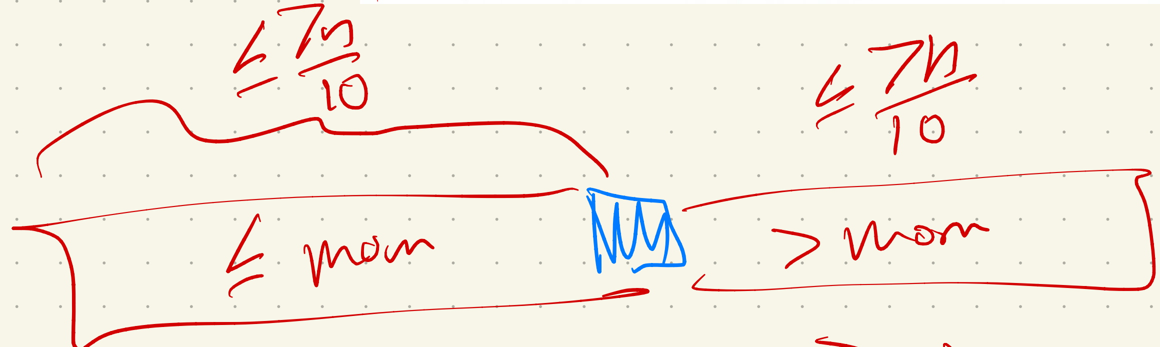
k

$O(n)$

First example of non-Master theorem!

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

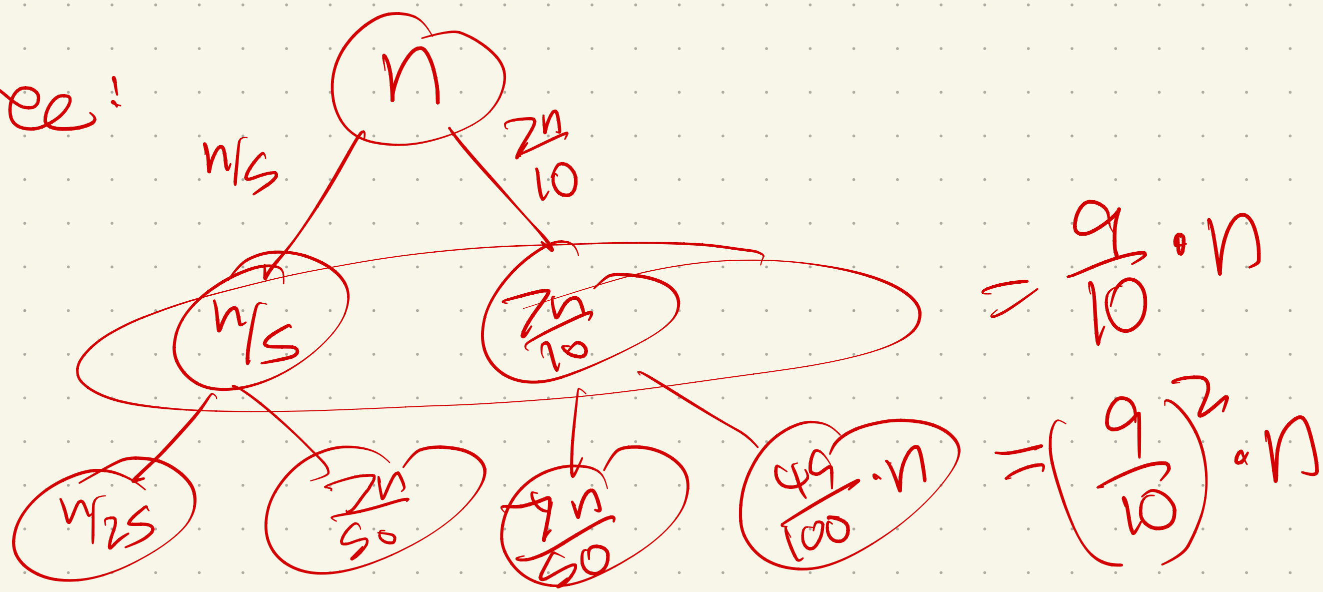




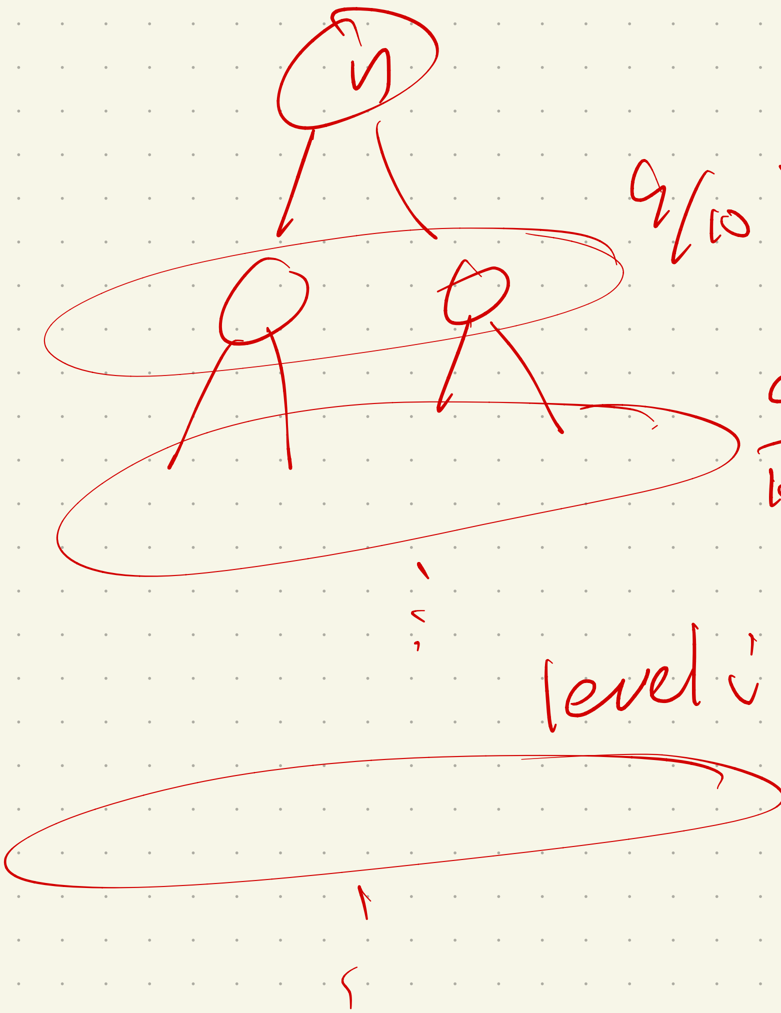
So:

$$M(n) \leq O(n) + M\left(\frac{n}{5}\right) + M\left(\frac{7n}{10}\right)$$

Tree!



Then solving:



$\frac{9}{10} \cdot n$

$\frac{9}{10}\left(\frac{9}{10} n\right)$

level i

$\left(\frac{9}{10}\right)^i \cdot n$

work total

depth d

$\left(\frac{9}{10}\right)^d \cdot n = 1$

$10/9$

$d = \log_{10/9} n$

$$\sum_{i=0}^{\log n} \left(\frac{9}{10}\right)^i \cdot n$$

$$= n \sum_{i=0}^{\log_{10/9} n} \left(\frac{9}{10}\right)^i$$

$$\le n \cdot \sum^{\infty} r^i \quad r < 1$$

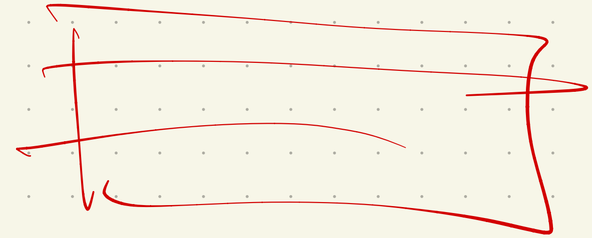
$$= n\left(\frac{1}{1-9/10}\right)$$

$$= 10 \cdot n$$

$$= O(n)$$

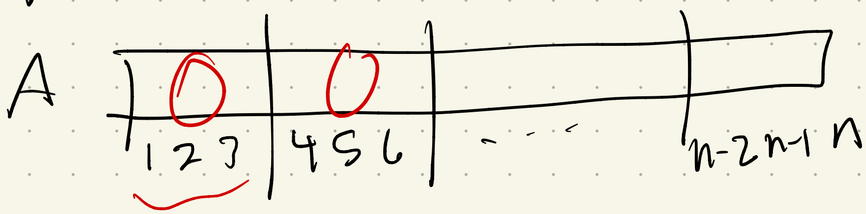
If did 3:

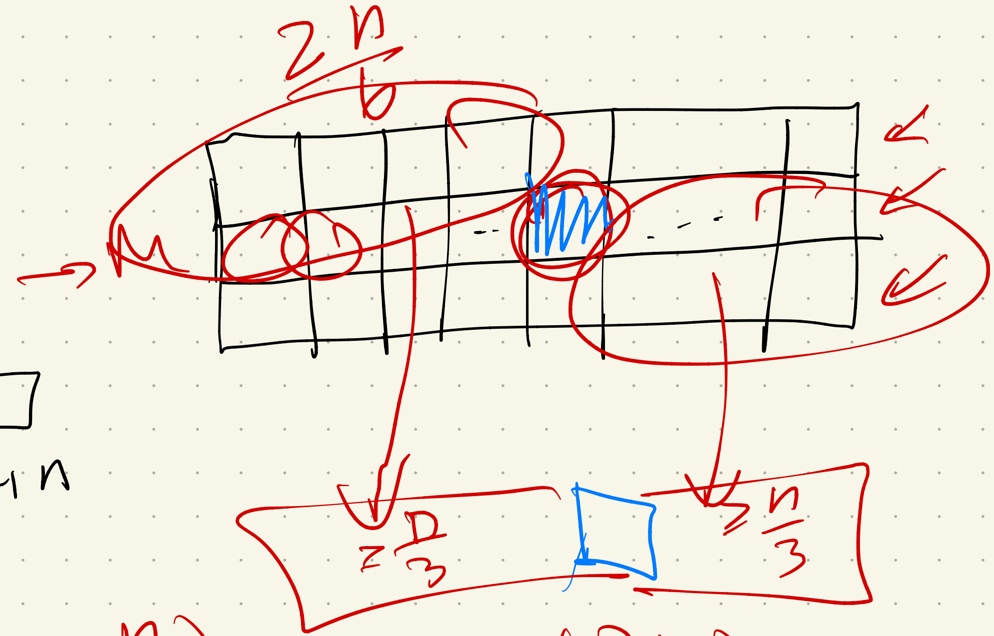
$T(n) \leq$



Why $\frac{n}{5}$ blocks?

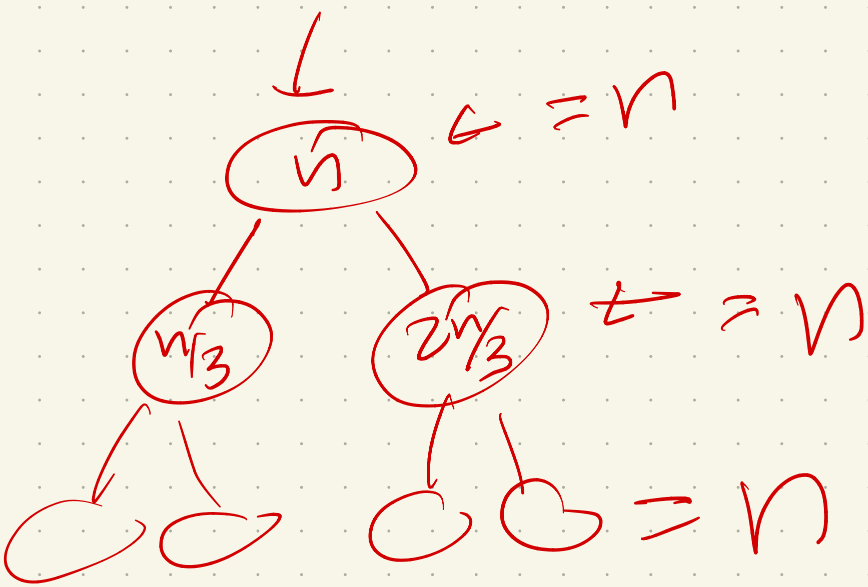
Try $n/3$ blocks:

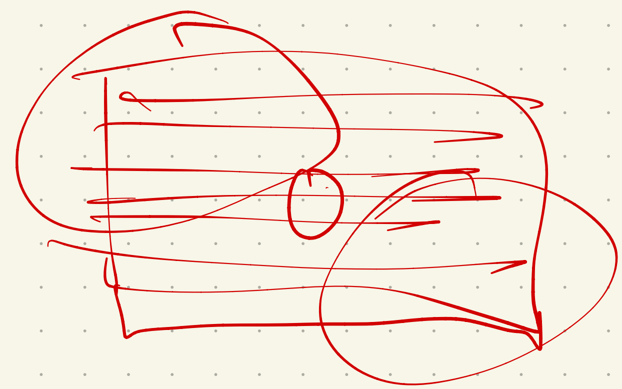




Result: $M(n) \le M(\frac{n}{3}) + M(\frac{2n}{3}) + O(n)$

$= n \log n$





Backtracking: N Queens

Issue:

representation!

$Q[1] = 5$

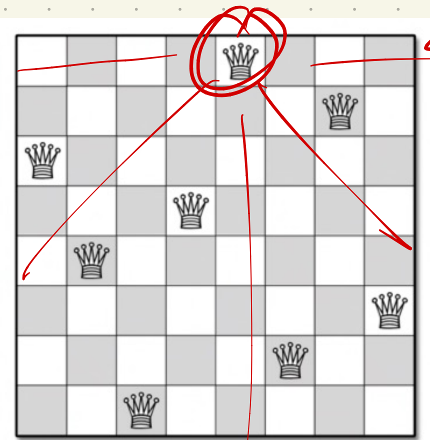


Figure 2.1. Gauss's first solution to the 8 queens problem, represented by the array $[5, 7, 1, 4, 2, 8, 6, 3]$

His choice: for each row, remember column #

$Q[1..n]$, each $Q[i]$ in $[1..n]$

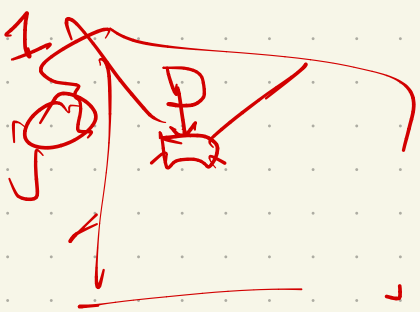
How to solve?

Structured brute force, set up recursively

Main tricky bit:

math to check

i = 1



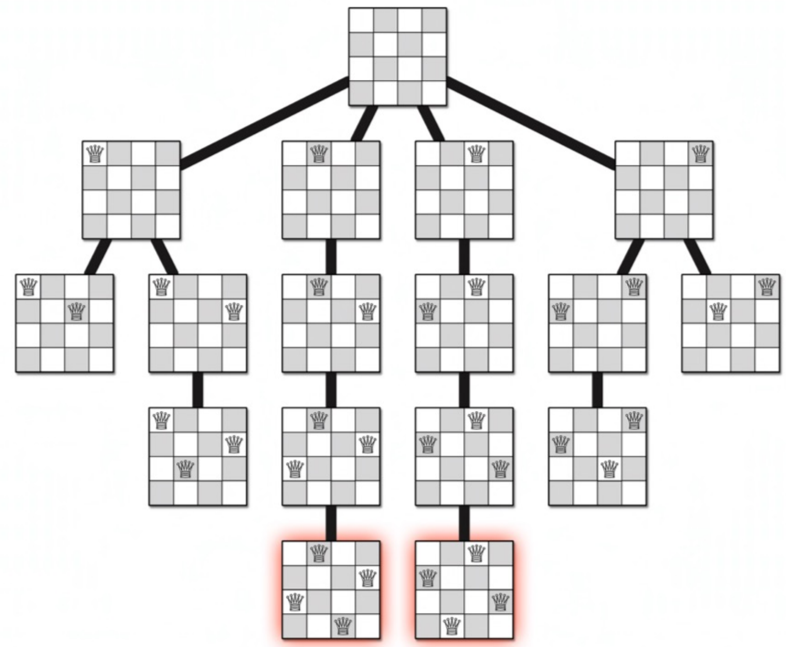


Figure 2.3. The complete recursion tree of Gauss and Laquière's algorithm for the 4 queens problem.

in same column

diagonal

```
PLACEQUEENS(Q[1..n], r):
  if r = n + 1
      print Q[1..n]
  else
      for j ← 1 to n
          legal ← TRUE
          for i ← 1 to r − 1
              if (Q[i] = j) or (Q[i] = j + r − i) or (Q[i] = j − r + i)
                  legal ← FALSE
          if legal
              Q[r] ← j
              PLACEQUEENS(Q[1..n], r + 1)        ⟨⟨Recursion!⟩⟩
```

Figure 2.2. Gauss and Laquière's backtracking algorithm for the n queens problem.

Runtime

$$Q(n) \leq nQ(n-1) + n^2$$

BAD

No way to improve

Game Trees:

a way to model moves in 2-player games

Assume:

- No randomness so the game is just 2 people taking turns

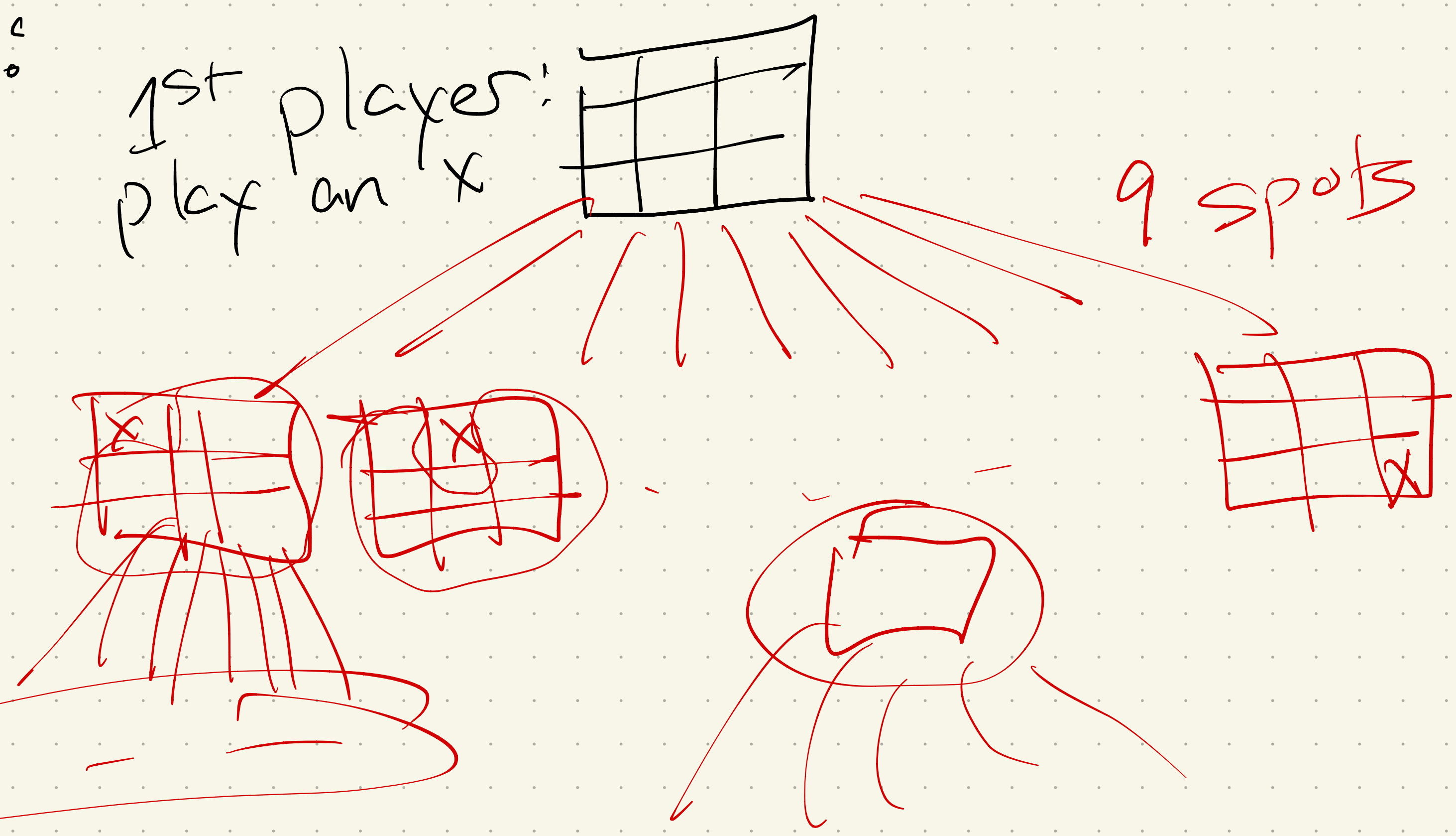
  Ex: Chess, Checkers, Nim, Go (not Settlers)

- "Perfect" players:

  Makes rational decisions, + if there IS a move to get them to a win state, they do it!

Idea: Track current state of the game, as play occurs

Tic-tac-toe:

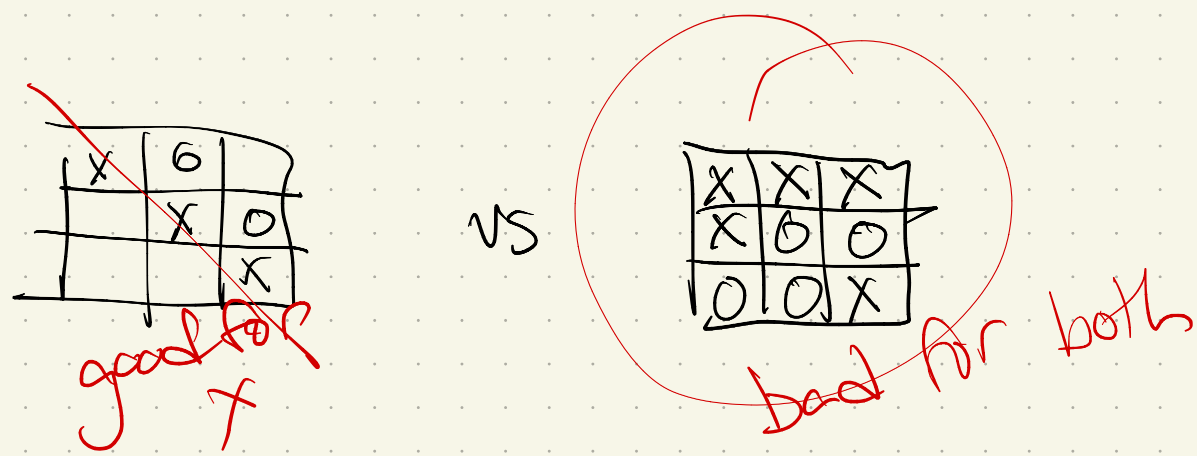


2nd player puts O

1st player again

⋮

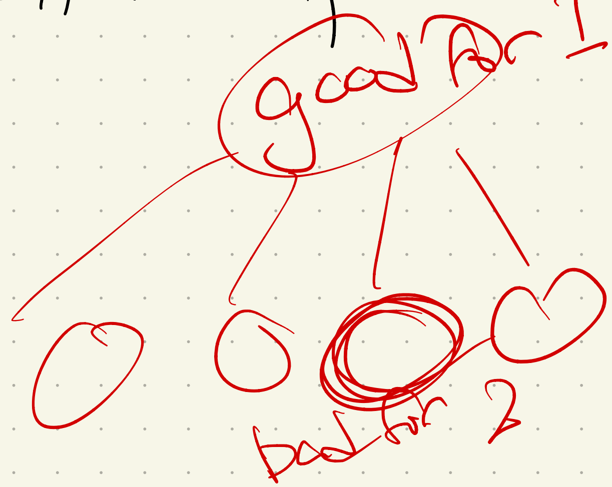
leaves: full, or someone wins:



A state is good for player 1 if they either have won, or could move to a bad state for player 2.

and bad if they have lost, or if all possible moves lead to a state that is good for player 2.

Think from the bottom up:



leaf: good if you win
bad if not

<u>Downsides</u>: Game trees are <u>HUGE</u>!

Tic-tac-to: over 200,000 leaves.

People can still "predict":
we're good at inferring state/strategy
intuitely

Computers have to search.
Hence - took 60 years to get a decent
computer chess player! Need
"heuristics" (aka guesses) to make it
work.

Game theory — a bit more complicated.

Here, we assume clear win vs. lose

← other course

Game theory suggests more subtle possibilities, as well as simultaneous moves & "randomness".

Example: Odds and Evens

Consider the simple game called **odds and evens**. Suppose that player 1 takes evens and player 2 takes odds. Then, each player simultaneously shows either one finger or two fingers. If the number of fingers matches, then the result is *even*, and player 1 wins the bet ($2). If the number of fingers does not match, then the result is *odd*, and player 2 wins the bet ($2). Eac[h] player has two possible strategies: show one finger or show two fingers. The *payoff matrix* shown below represents the payo[ff] to player 1.

Payoff Matrix

Strategy		Player 2: 1	Player 2: 2
Player 1	1	2	-2
	2	-2	2

Even if we know all results, outcome is unclear!

Text Segmentation

↳ Leads well into next reading

Fix a "language", so can recognize "words".

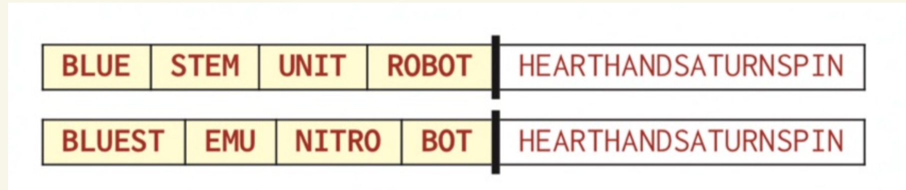
<u>Ex</u>: - English text
- Genetic data
⋮

So: Isword(s) is given, + O(1) time.

Aside: reasonable?

Backtracking:

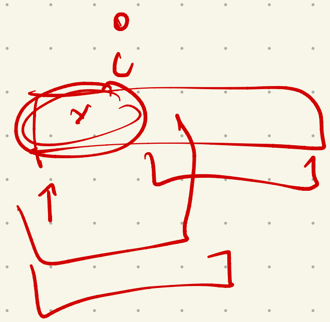
Fix suffix
to decide on.



To solve Splittable[i..n]:

Code:

```
SPLITTABLE(A[1..n]):
  if n = 0
      return TRUE
  for i ← 1 to n
      if ISWORD(A[1..i])
          if SPLITTABLE(A[i+1..n])
              return TRUE
  return FALSE
```



Runtime:

Issue w/ passing arrays:

Passing by index / ptr / global / etc

Given an index i, find a segmentation of the suffix $A[i\,..\,n]$.

Formalize an (ugly?) recursion:

$$Splittable(i) = \begin{cases} \textsc{True} & \text{if } i > n \\ \displaystyle\bigvee_{j=i}^{n} \big(\textsc{IsWord}(i,j) \wedge Splittable(j+1)\big) & \text{otherwise} \end{cases}$$

And then translate to code:

```
⟨⟨Is the suffix A[i..n] Splittable?⟩⟩
SPLITTABLE(i):
  if i > n
      return TRUE
  for j ← i to n
      if ISWORD(i, j)
          if SPLITTABLE(j + 1)
              return TRUE
  return FALSE
```

Why??

It's already exponential anyway, right?

Observation:

```
⟨⟨Is the suffix A[i..n] Splittable?⟩⟩
SPLITTABLE(i):
    if i > n
        return TRUE
    for j ← i to n
        if ISWORD(i, j)
            if SPLITTABLE(j + 1)
                return TRUE
    return FALSE
```

Consider stack point of view, + all of these function calls:

So: For any $k \in [1..n]$, might be calling Splittable(k) many times!

Question: Can its value change?
(ie is it a pure function?)

Potential Improvement

Once you calculate Splittable(k) once, store it.

Then, can just look it up in a data structure! S[1..n]

Here:

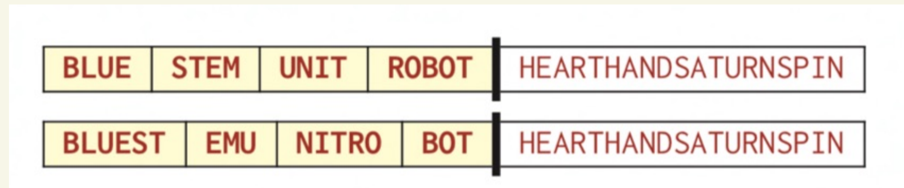
Then:

```
⟨⟨Is the suffix A[i..n] Splittable?⟩⟩
SPLITTABLE(i):
  if i > n
      return TRUE
  for j ← i to n
      if ISWORD(i, j)
          if SPLITTABLE(j + 1)
              return TRUE
  return FALSE
```

Change:

Better yet:

- Splittable(n) is trivial
- Splittable(n-1) only needs Splittable(n)
- Splittable(n-2) only needs n-1 & n-2



So: memoize & fill in backwards!

At end: return Splittable[1]