Algorithms

Recursion
Backtracking

Recap

- Usual Reading
- HW1 due Wednesday

Next: how to generalize?

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

← Master thm

What it means:

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

total $f(n)$ ops

$f(n)$

← r children

n/c

$f\left(\frac{n}{c}\right)$

… … $f\left(\frac{n}{c^i}\right)$ … level i: r^i nodes

Solving: those trees!

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

nodes on level i

work in each node

Ex: $\frac{n}{2^i}$ in MS

$\left(\frac{n}{2^i}\right)^2$ in other once

depth: $\log_c n = O(\log n)$

3 possibilities:

- decreasing geom. series (like my example) $f(n)$ will dominate
- increasing: # nodes dominates $n^{\log_c r}$

  ie: binary search! $B(n) = B(\frac{n}{2}) + O(1)$

- balanced — like merge sort

Master Thm:

$n^{\log_2 4} = n^2$

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = \underset{4}{a}T(n/\underset{2}{b}) + \underset{O(1)}{f(n)},$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ∎

Recurrence: Master Theorem

T(n) = aT(n/b) + f (n) where f (n)=cn^k

1. $a < b^k$ $\quad T(n) \sim n^k$
2. $a = b^k$ $\quad T(n) \sim n^k \log_b n$
3. $a > b^k$ $\quad T(n) \sim n^{\log_b a}$

$$T(n)=\begin{cases}\Theta\left(n^{\log_b a}\right) & f(n)=O\left(n^{\log_b a-\varepsilon}\right)\\ \Theta\left(n^{\log_b a}\log n\right) & f(n)=\Theta\left(n^{\log_b a}\right)\\ \Theta\left(f(n)\right) & f(n)=\Omega\left(n^{\log_b a+\varepsilon}\right)\text{AND}\\ & af(n/b)<cf(n)\text{ for large } n\end{cases}\quad \begin{matrix}\varepsilon>0\\ c<1\end{matrix}$$

Other examples

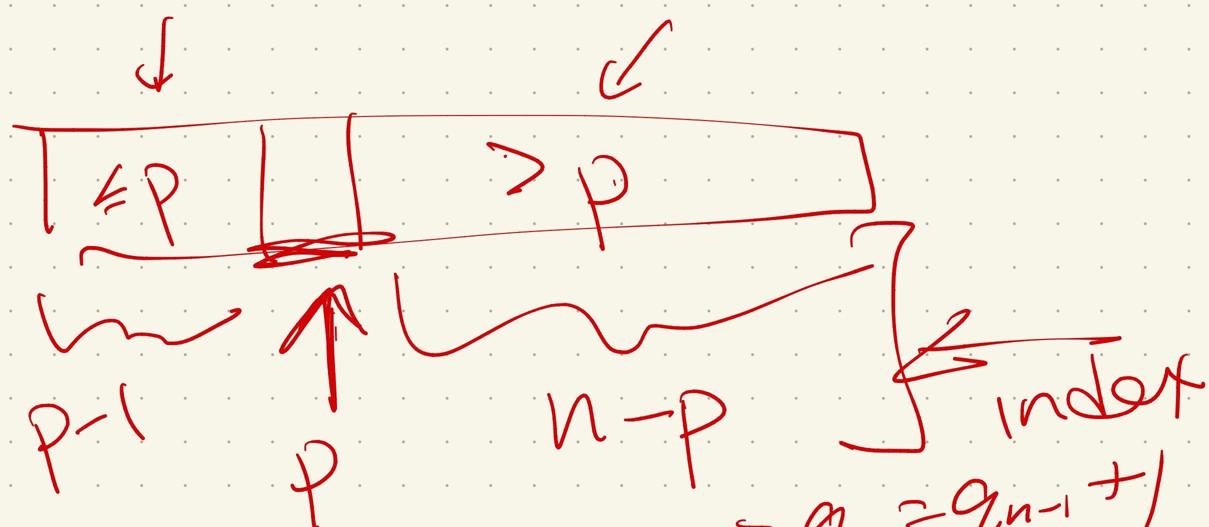
Medians: find "middle" element. $k = \frac{n}{2}$

Two were covered: looking for any k

```
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



$k < p-1$

$k > p$

$q_n = q_{n-1} + 1$

$Q(n) \leq Q(n-1) + n$

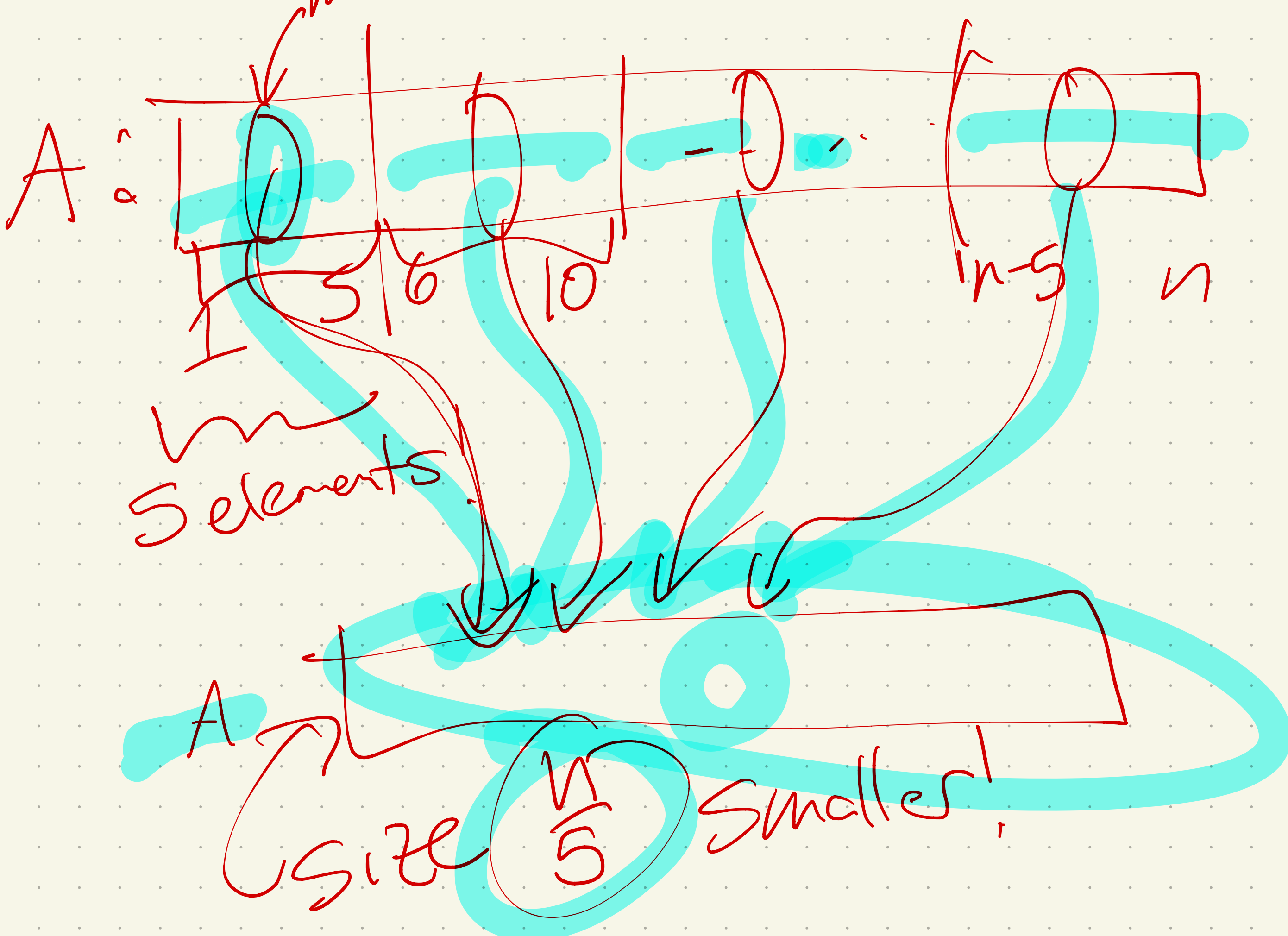
"Faster" version:

Q: why 5?

```
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
```

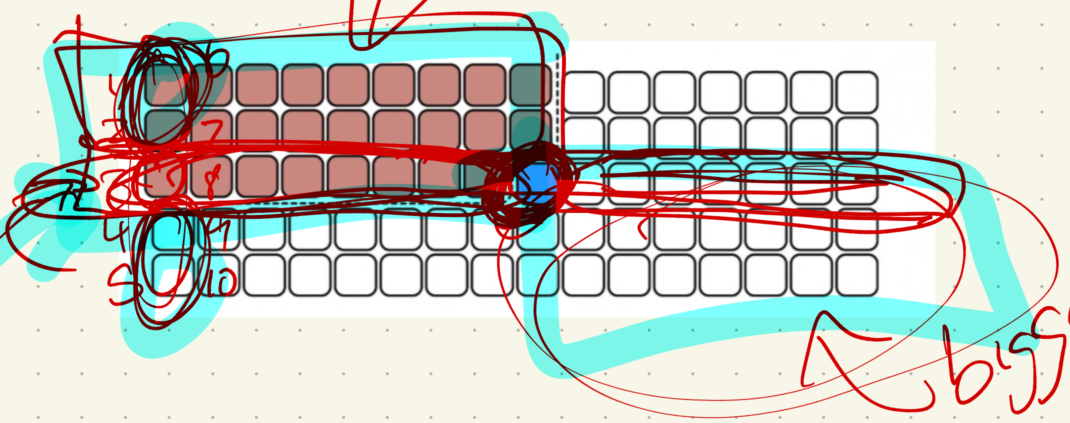
$O(n)$

$\frac{n}{5}$



this picture:

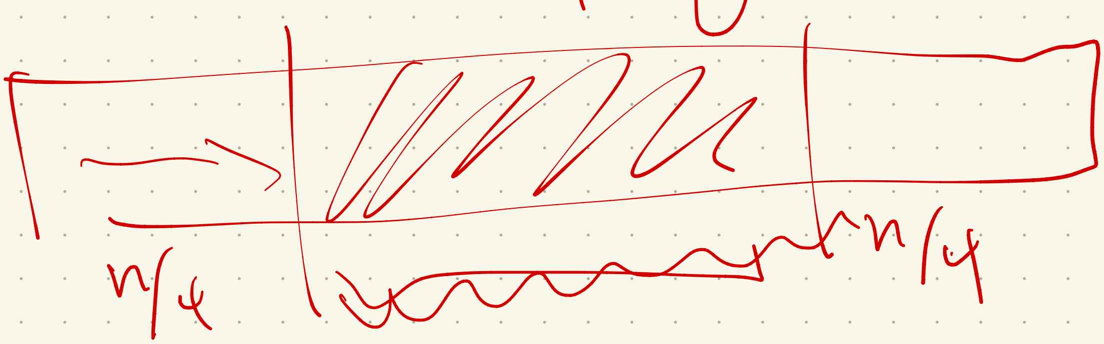
≤ blue cell



bigger than blue medians

that array of n/5 medians

Result: temporary median, not perfect, but with $\frac{n}{4}$ elements less & $\frac{n}{4}$ greater



Runtime:

```
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
```

base case: $O(1)$

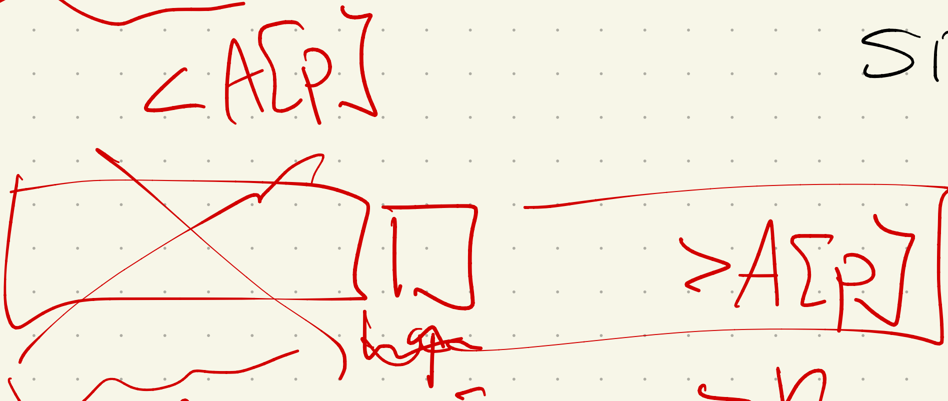
$O(1)$

loop: $O(n)$

$\frac{n}{5} \cdot O(1)$

recursive call, size $= \frac{n}{5}$

another recursive call, size $\leq$



$\geq \frac{n}{4}$

$f(n) \geq \frac{n}{4}$

$c = 5$

$c = 4/3$

$$M(n) \leq O(n) + M\left(\frac{n}{5}\right) + M\left(\frac{3n}{4}\right)$$

↪ can't use Master theorem!

$$n$$

$$\frac{n}{5} \qquad \frac{3n}{4}$$

$$\frac{n}{5} \qquad \frac{3n}{4} = \left(\frac{4}{20} + \frac{15}{20}\right) n = \frac{19}{20} \cdot n$$

$$\frac{19}{20} < 1$$

M[size $\frac{n}{5}$]

$$\frac{n}{25} \qquad n\left(\frac{1}{5}\right)\left(\frac{3}{4}\right) \qquad n \cdot \frac{1}{5} \cdot \frac{3}{4} \qquad \frac{n}{\left(\frac{4}{3}\right)^2} \qquad \left(\frac{19}{20}\right)^2 \cdot n$$

depth $= O(\log n)$

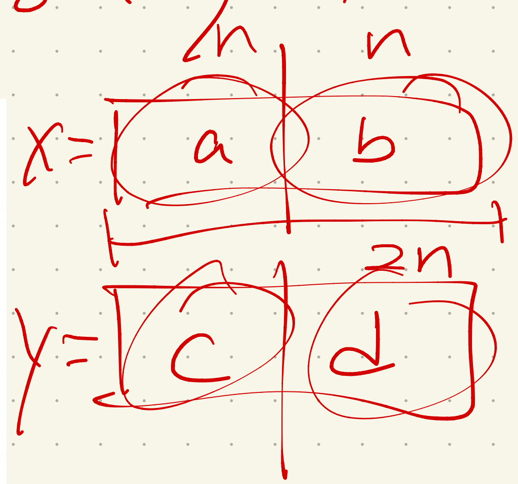
$\log_5 n$ $\qquad$ $\log_{4/3} n$

$$\sum^{\log n} \left(\frac{19}{20}\right)^i \cdot n$$

$$= \boxed{O(n)}$$

Multiplication: factoring trick!

```
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
```

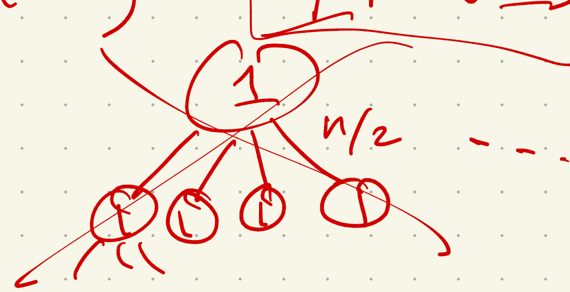


$$xy = \underbrace{(a\cdot 10^n + b)}_{= x}\underbrace{(c\cdot 10^n + d)}_{= y}$$

Runtime:

apply MT

$$M(n) = 4M\left(\frac{n}{2}\right) + O(1) = O\left(n^{\log_2 4}\right) = O(n^2)$$



VS.

```
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
```

$$(a10^n + b)(c10^n + d) = 10^{2n}ac + \dots$$

algebra trick

Runtime: $F(n) = 3F\left(\frac{n}{2}\right) + O(1)$

apply MT: $O(1)$ vs $n^{\log_2 3}$ $\Rightarrow O\left(n^{\log_2 3}\right)$

Exponentiation: compute # of multiplications to get a^n

can do n

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!

Ch 2: Backtracking:

Many of you saw in AI, apparently!

(Don't worry if not...)

Why we discuss:

It's really recursion (again)!

Also really a form of brute force:

try everything recursively, & see what works.

N Queens

row

A[i] = column number of queen in row i

queen per row + col. but no 2 diagonal

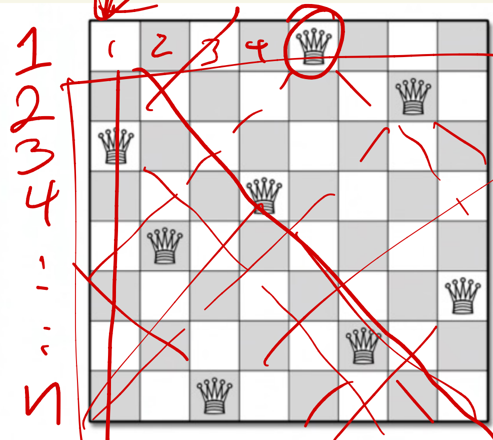


Figure 2.1. Gauss's first solution to the 8 queens problem, represented by the array [5, 7, 1, 4, 2, 8, 6, 3]

Issue: representation!

His choice: one per row, so store index of queen on rows in array.

Now, how to solve:

Brute force! Place a queen + keep going. If you get stuck, "unplace" last queen + back up.

The tree (b/c pretty) ◡̈

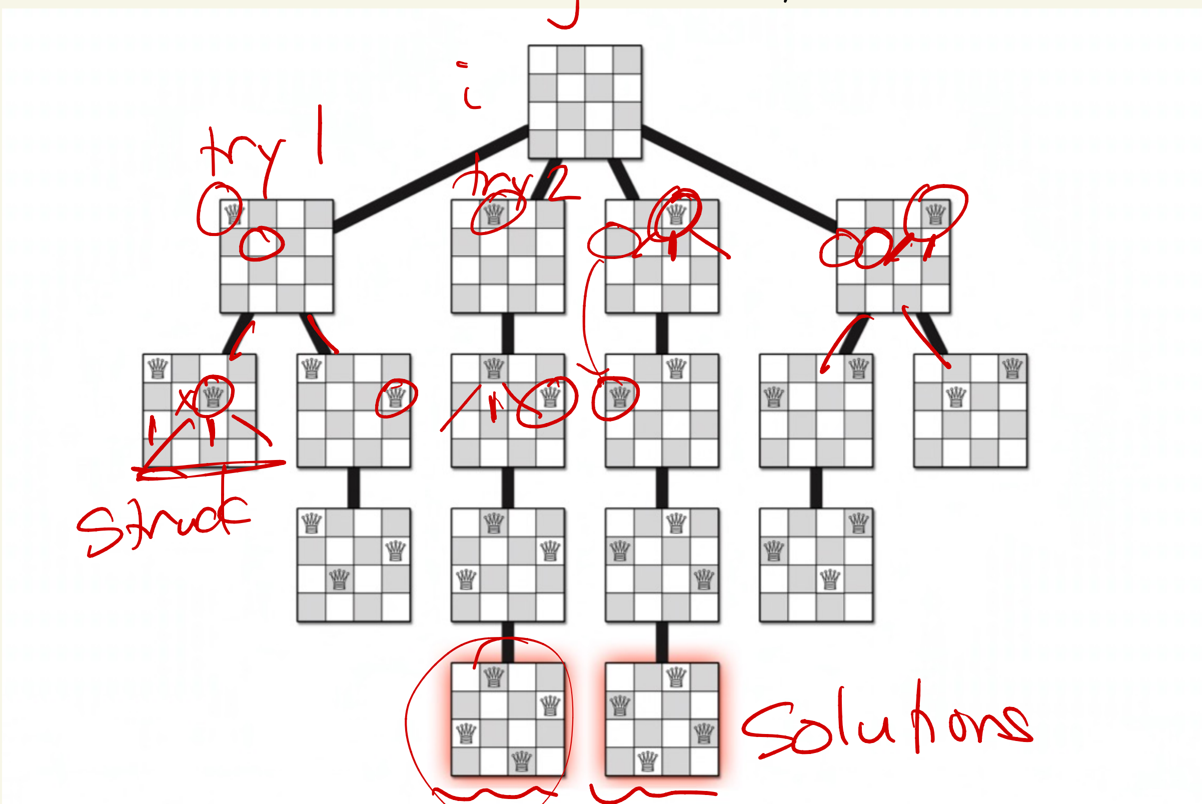


Figure 2.3. The complete recursion tree of Gauss and Laquière's algorithm for the 4 queens problem.

Problem (& hard part):

Formalizing this in code.

Sketch: try each pos j & recurse on viable next choises

Result:

```
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:

$$Q(n) =$$