Recursion

Seen recurrences:

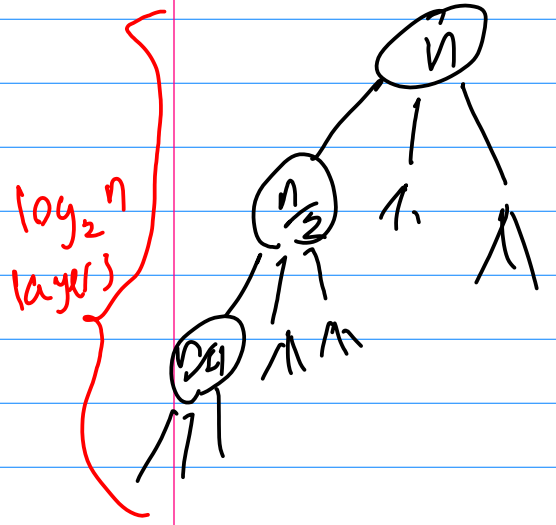
Multiplication (Karatsuba)

$$T(n) = 3T(n/2) + O(n)$$

Merge Sort:

$$T(n) = 2T(n/2) + O(n)$$

Solving recurrences (way 1): draw tree



(# nodes)	(work per node)	(total at level)
1	n	n
3	$\frac{n}{2}$	$(\frac{3}{2})n$
3^2	$\frac{n}{4}$	$(\frac{3}{2})^2 n$
⋮	⋮	⋮
$3^{\log_2 n} = n^{\log_2 3}$	1	$n^{\log_2 3}$

↳ Geometric series! Sum is $O(n^{\log_2 3})$

Geometric Series

Let $S = a + ar + ar^2 + \cdots + ar^n + \cdots$ $(r < 1)$

$(1-r)S = a + ar + ar^2 + \cdots$
$\quad\quad - ar - ar^2 - \cdots$

$$S = \frac{a}{1-r}$$

So: $S = a + ar + \cdots + ar^{k-1}$ is:

$\Theta(a)$ if $r < 1$ a constant

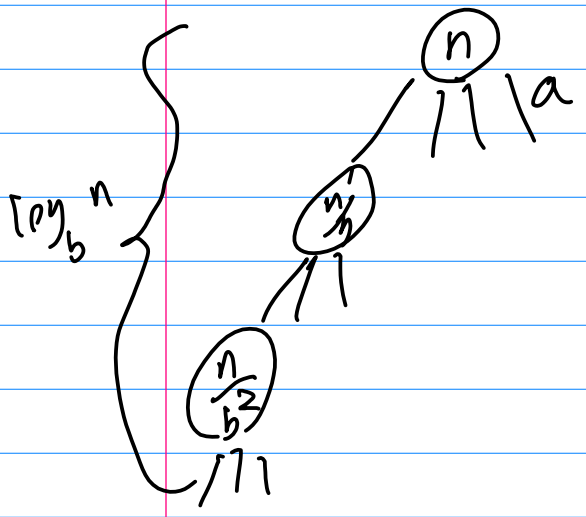
($r = \frac{3}{4}$ or 0.9 not $1 - \frac{1}{a}$)

$\Theta(ar^{k-1})$ if $r > 1$ a constant

$\Theta(a \cdot k)$ if $r = 1$

(bigger of first and last, with an extra factor k if equal)

$$T(n) = a\,T\left(\frac{n}{b}\right) + n^c$$



	# nodes	work per node	total
	1	n^c	n^c
	a	$\left(\frac{n}{b}\right)^c$	$\left(\frac{a}{b^c}\right)n^c$
	$\vdots$	$\vdots$	$\vdots$
	a^k	$\left(\frac{n}{b^k}\right)^c$	$\left(\frac{a}{b^c}\right)^k n^c$
	$\vdots$	$\vdots$	$\vdots$
	$a^{\log_b n} = n^{\log_b a}$	1	$n^{\log_b a}$

geometric series w/ ratio $r = \frac{a}{b^c}$.

Define

$$c_{crit} := \log_b a.$$

$c < c_{crit} \Rightarrow r > 1 \Rightarrow$ dominated by $n^{\log_b a}$

$c > c_{crit} \Rightarrow r < 1 \Rightarrow$ dominated by n^c

$c = c_{crit} \Rightarrow r = 1 \Rightarrow$ total is $\mathcal{O}(n^c \log_b n)$

$= \mathcal{O}(n^c \log n)$

Master Theorem generalizes this slightly:

$T(n) = a\,T(\frac{n}{b}) + f(n)$:

(1) if $f(n) = O(n^c)$ for $c < c_{crit}$

$T(n) = \theta(n^{\log_b a})$

(2) if $f(n) = \theta(n^{c_{crit}} \cdot \log^k n)$ for $k \geq 0$

[e.g. $n^{c_{crit}}$ or $n^{c_{crit}} \log n$]

$T(n) = \theta(n^{c_{crit}} \log^{k+1} n) = \theta(f(n) \log n)$

(3) if $f(n) = \Omega(n^c)$ for $c \geq c_{crit}$

- AND REGULARITY CONDITION

$T(n) = \theta(f(n))$

* TECHNICAL NOTE: REGULARITY

- Holds if $f(n) = \theta(n^c \log^k n)$ for some c, k

- It's just that work ratio

$r = a\,f(n/b)/f(n) < 1 \quad \forall\, n.$

- Needed to avoid pathological cases, e.g.

Using master theorem:

multiplication:

$c_{crit} = \log_2 3$, $c = 1 < c_{crit} \Rightarrow$ case (1)

dominated by last layer $\rightarrow$ $T(n) = \Theta(n^{\log_2 3})$

merge sort

$c_{crit} = \log_2 2 = 1$, $c = 1 = c_{crit} \Rightarrow$ case (2)

balanced $\rightarrow$ $T(n) = \Theta(n \log n)$

$T(n) = 3T(n/2) + n^2$

$c_{crit} = \log_2 3$, $c = 2 > c_{crit} \Rightarrow$ case (3)

dominated by first layer $\rightarrow$ $T(n) = n^2$

Recursive Backtracking

Easy way to solve programs Correctly but Slowly

Next weeks (DP): techniques to make fast

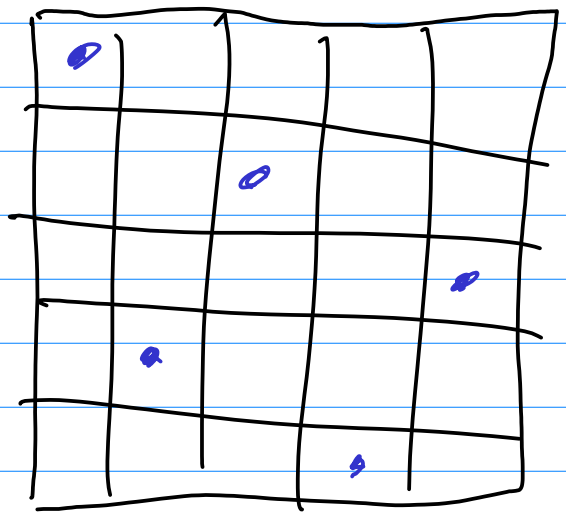
Idea: to find
- a solution
- the best solution
- the # of solutions
- the smallest solution

to some problem,

represent a solution as a series of choices,
try all possibilities for one choice in function body,
recurse on rest.

n queens

Place n queens on n x n board so no two attack each other.



Choices:
- location in row 1, in row 2, ...

columns where queen placed

```
Place Queens (Q, n):
    if len(Q) == n: return Q
    r = len(Q)
    for j in range(n):   # try placing (r, j)

        if any( i == j   or
                abs(i-j) == r-s
                for s, i in enumerate(Q) ):

            Continue
        attempt = placeQueens(Q+[j], n)
        if attempt:
            return attempt
```

Subset Sum:

given $x_1, \ldots, x_n$ integers

Target T

Does $\exists$ subset $S \subseteq [n]$ s.t. $\sum_{i \in S} x_i = T$

Choices: take x_1, x_2, ...

```
def SubsetSum(X, T)
    if len(X) == 0:
        return T == 0
    v = X[0]
    return SubsetSum(X[1:], T)        # don't take v
        or SubsetSum(X[1:], T - v)    # take v
```

Can't slower since copy inputs/vars, can fix)

```
def SubsetSum(X, T, i):
    if len(X) == i:
        return T == 0
    v = X[i]
    return SubsetSum(X, T, i+1)       # don't
        or SubsetSum(X, T - v, i+1)   # take
```

Largest Increasing Subsequence:

$A_1, \longrightarrow, A_n$

find largest subset where A_i increasing

Different ways to express as series of choices:

Take A_1? Take A_2? Take A_3?

$$LIS(A) = \begin{cases} 0 \text{ if len}(A) = 0, \text{ else} \\ \max(LIS(A[1:]), \\ \quad 1 + LIS([x \text{ for } x \text{ in } A[1:] \text{ if } x \geq A[0]]) \end{cases}$$

don't take first

take first

OR Choice is "who to take first?" "Who to take second?" ...

$$LIS(A) = \begin{cases} 0 \text{ if len}(A) = 0, \text{ else} \\ \max_i (1 + LIS([x \text{ for } x \text{ in } A[i+1:] \text{ if } x \geq A[i]])) \end{cases}$$

Different expression, same result.