Announcements

Announcements

- Homework 1 is out!
- Due February 23 (one week) at midnight anywhere on Earth

Recurrence Relations

Recurrence Relations

not pseudocode!

- A recurrence relation is an equation that recursively defines a function's values in terms of earlier values
- Very useful for analyzing an algorithm's running time!

Recurrence Relations in Code

```
def betterPower(x, n):
    if n == 0:
        return 1
    else if n == 1:
        return x
    else if n % 2 == 0:
        return betterPower(x * x, n/2)
```

$T(n)$ = running time for bP on value n (for arbitrary x)

if $n = 0$: $T(0) = c_1$

if $n = 1$: $T(1) = c_2$

if $n > 1$: $T(n) = c_3 + T(n/2)$

(assume for simplicity that n is a power of 2)

How can we write the running time?

Recurrence Relations in Code

$T(0) = c_{1,}\ \ T(1) = c_2,\ \ \ldots,\ T(n) = c_3 + T(n/2)$

$$
\begin{aligned}
T(n) &= c_3 + (T(n/2)) \\
&= c_3 + (c_3 + T(n/4)) \\
&= c_3 + c_3 + (c_3 + T(n/8))
\end{aligned}
$$

.....

$$
\begin{aligned}
&= kc_3 + T(n/(2^k)) \\
&= c_3 * \log n
\end{aligned}
$$

$+ c_1$

What should k be in order for us to get down to T(1)?

$2^k = n$, so $k = \log n$

Solving Recurrences

- Solving recurrence relations is like integrating an expression- there are tricks, but no techniques are guaranteed to work

Solving Recurrences

- Simplest method: Guess the solution, prove with induction
- Sometimes it helps to do a few expansions to get some intuition

Bounding Recurrences: the Master Method

"other work" besides recursion

Suppose $T(n) = aT(\lceil \frac{n}{b} \rceil) + O(n^d)$,

for $a > 0$, $b > 1$, $d \geq 0$.

constants

"ceiling" symbol: means to round up

$O(\log_a x) = O(\log_b x)$

- Case 1: If $d > \log_b a$, then $T(n) = O(n^d)$
- Case 2: If $d = \log_b a$, then $T(n) = O(n^d \log n)$
- Case 3: If $d < \log_b a$, then $T(n) = O(n^{\log_b a})$

no base

Note that this doesn't actually SOLVE the recurrence!

Example

$T(1) = ?$ $O(n^3)$

Bound the recurrence $T(n) = {}^{a}1\,T\left(\left\lceil \frac{n}{{}_{b}2} \right\rceil\right) + 3n^3 + 2.$

Parameters:

a = 1

b = 2

d = 3

Case = $\log_2 1 = 0 < 3$

Solution =

case 1 $T(n) = O(n^3)$

$T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$
• Case 1: If $d > \log_b a$, then $T(n) = O(n^d)$ • Case 2: If $d = \log_b a$, then $T(n) = O(n^d \log n)$ • Case 3: If $d < \log_b a$, then $T(n) = O(n^{\log_b a})$

Example: Binary search

$O(n^d)$ = "other work" ignoring recursion

a = number of rec. calls generated by each function call

Algorithm: Goal is to find the location of a value in a sorted array. Start by looking in the middle; then depending on the value there, look in either the first half or second half, and so on. Divide the array in half each time.

k=3

b = size of input in the rec. call

[1 3 4 6 8 9 10]

a=1

b=2

What is the running time of binary search?

T(n) = worst case running time of BinSearch on an array of size n

$$T(n) = 1 \cdot T\left(\frac{n}{2}\right) + O(n^0)$$

Example: Binary search

if other work = $O(n^1)$, then $d=1$

Write the running time as a recurrence relation:

What is the running time?

a = 1

b = 2

d = 0 $O(1) = n^0$

Case = $\log_2 1 = 0 = d$

Solution = $O(n^0 \log n)$

case 2 = $O(\log n)$

BinSearch(A, k)
find mid
compare mid to k
recurse on either left or right half

N
N/2
N/4
⋮
1

Example: Binary search

[1 3 4 6 8 9 10]

BinSearch(A, k)

if len(A) == 1: if A[0] == k, return that index

else:

get midpoint ← constant time because array

compare mid to k ← constant time

decide which half to branch on

create subarray representing that half

→ recurse on that subarray

branch

In-Class Exercise

$\log n = O(n)$ $\quad O(n^0) = O(1) = \text{constant}$

$T(n) = 3T(\frac{n}{2}) + \log n \quad a = 3,\ b = 2 \quad d = 1$

Bound the following recurrences:

1. $T(n) = T(\lceil \frac{n}{3} \rceil) + 2n^2 + 2$

   $a = 1 \quad b = 3 \quad d = 2$ $\quad$ C1 $T(n) = O(n^2)$

2. $T(n) = 5T(\lceil \frac{n}{2} \rceil) + 5n^4$

   $a = 5 \quad b = 2 \quad d = 4$ $\quad T(n) = O(n^4)$

3. $T(n) = 2T(\lceil \frac{n}{2} \rceil) + 3n^3 + 2n$

   $a = 2 \quad b = 2 \quad d = 3 \quad O(n^3)$

4. $T(n) = 3T(\lceil \frac{n}{2} \rceil) + 2^n$

   $a = 3 \quad b = 2 \quad d = \cancel{n}\,?$

   $2^n \neq O(n^d)$ for constant d

   MM does not apply

$$T(n) = aT(\lceil \frac{n}{b} \rceil) + O(n^d)$$

- Case 1: If $d > \log_b a$, then $T(n) = O(n^d)$
- Case 2: If $d = \log_b a$, then $T(n) = O(n^d \log n)$
- Case 3: If $d < \log_b a$, then $T(n) = O(n^{\log_b a})$

Proof of the Master Method

For simplicity, we assume that n is a power of b. This does not affect the final result, because n is at most a constant factor of b away from a power of b.

$$T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$$

$\uparrow$

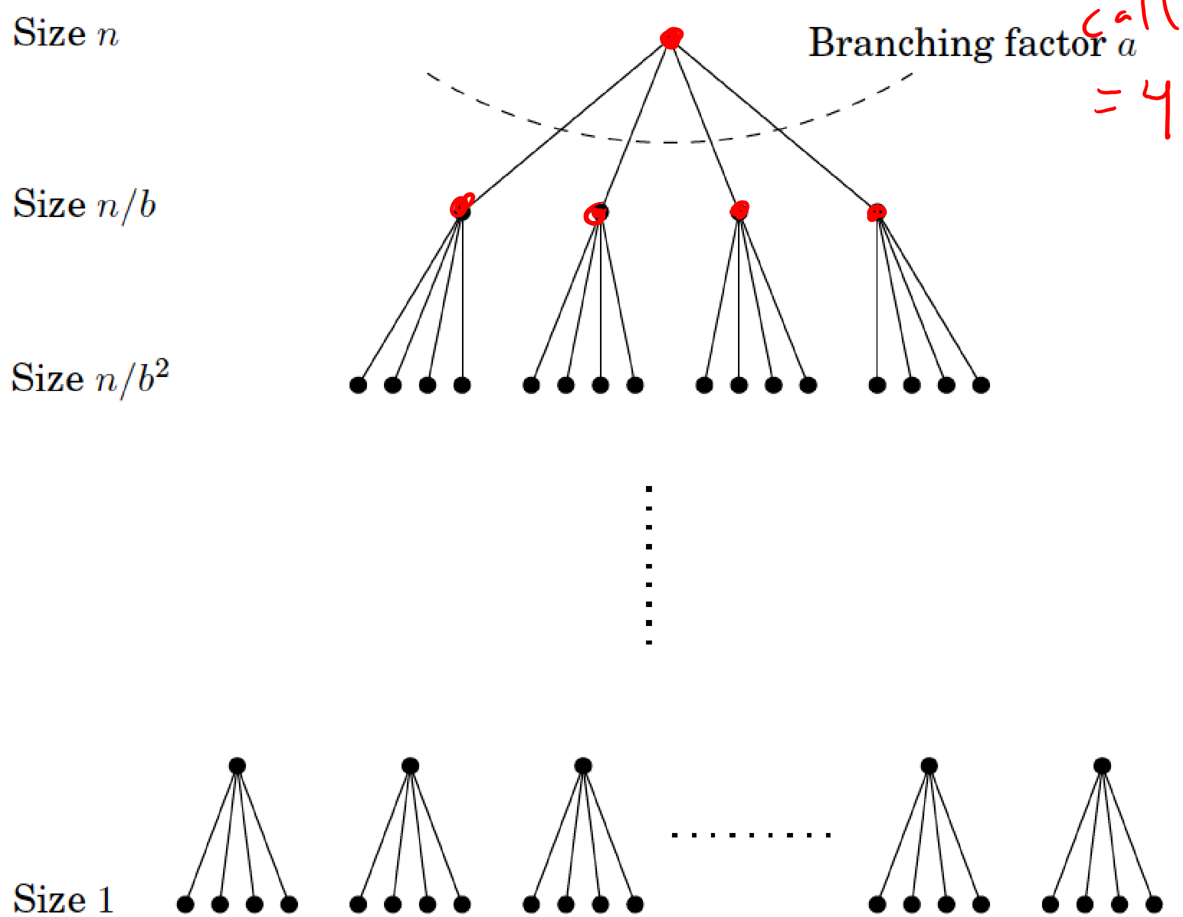
Recursion Tree

#rec. calls

other work

$$T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$$

size of each call



Recursion Tree

function (A)
→ some work
→ some recursion

$b^? = n$

work assoc. w/ each call

$$T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$$

not same for all nodes

\# Levels?

$\log_b n$

Problems in level *k*?

(root = L0) a^k

Size of problem in level *k*?

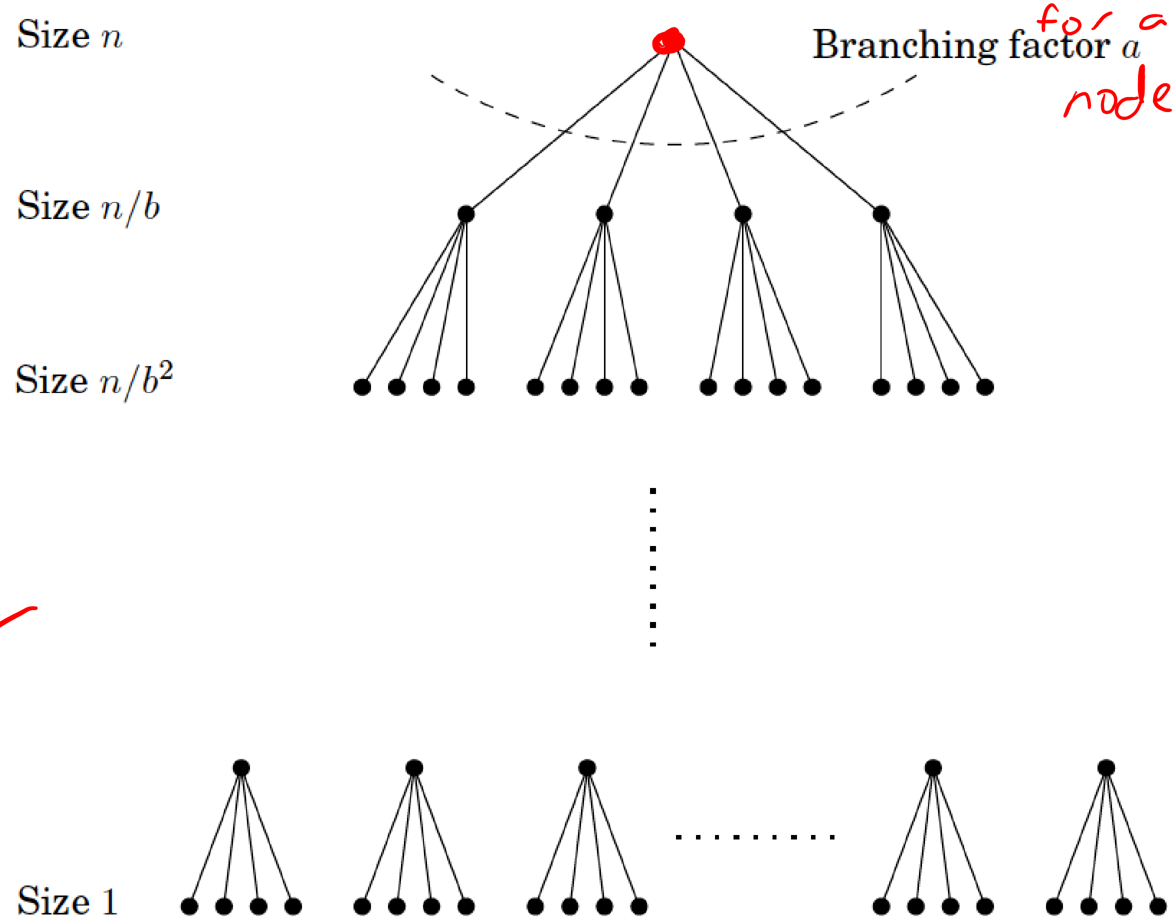
$\frac{n}{b^k}$

Work done in each problem? in level k

$O\left(\left(\frac{n}{b^k}\right)^d\right)$

Work done in level *k*?

$O\left(a^k \left(\frac{n}{b^k}\right)^d\right)$



Proof of the Master Method

$$\sum_{k=0}^{\log_b n} O\left(a^k \left(\frac{n}{b^k}\right)^d\right) = \text{total running time}$$

$$\sum O\left(a^k \cdot \frac{n^d}{b^{kd}}\right) = n^d \sum_{k=0}^{\log_b n} O\left(\left(\frac{a}{b^d}\right)^k\right)$$

$$r = \left(\frac{a}{b^d}\right) \qquad = n^d \sum_{k=0}^{\log_b n} O(r^k)$$

Proof of the Master Method

Case 1: $r<1$, ratio decreases as k goes up
sum is given by $O(n^d \cdot r^0) = O(n^d)$

Case 2: $r=1$, then $r^k = 1$, so the sum
is given by $O(n^d \cdot \#\,\text{terms}) =$
$O(n^d \log_b n) = O(n^d \log n)$

Proof of the Master Method

Case 3: $r > 1$, r^k increases as k goes up, sum dominated by last term

$$O(n^d \cdot r^{\text{highest } k}) = O(n^d \cdot r^{\log_b n})$$

$$= O\left(n^d \cdot \left(\frac{a}{b^d}\right)^{\log_b n}\right) = O\left(n^d \cdot \frac{a^{\log_b n}}{(b^{\log_b n})^d}\right) =$$

$$= O(a^{\log_b n}) = O(n^{\log_b a}).$$

Proof of the Master Method

MM: d vs. $\log_b a$

proof: compared $r = \frac{a}{b^d}$ vs. 1

In-Class Exercise

T(n) = running time on input of size n

Analyze the running time of the following function:

$T(n) = 3T(\frac{n}{3}) + O(n^1)$

x = 0

* for i = 1:len(A)

x += 1

O(n)

```
function SplitMax(array A):
    if length(A) == 1:
        return A[0]
    else:
        A1 = A[0 : length(A)/3]
        A2 = A[(length(A)/3) + 1 : 2*(length(A)/3)]
        A3 = A[2*(length(A)/3) + 1 : length(A)]
        return max(SplitMax(A1), SplitMax(A2), SplitMax(A3))
```

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

$\frac{n}{3}$

O(1)

O(1)