Recurrences

David Kauchak
cs302
Spring 2012

Administrative

- Talk today
- Assignment 1
  - for proofs by induction, make sure you make the steps clear:
    - base case
    - inductive case
      - assumption (inductive hypothesis)
      - what you're trying to prove
      - proof
- Assignment 2?
- Assignment 3 out today
- Latex?
- My view on homework…

MergeSort

```
MERGE-SORT(A)
if length[A] == 1
        return A
else
        q ← ⌊length[A] /2⌋
        create arrays L[1..q] and R[q + 1.. length[A]]
        copy A[1..q] to L
        copy A[q + 1.. length[A]] to R
        LS ← MERGE-SORT(L)
        RS ← MERGE-SORT(R)
        return MERGE(LS, RS)
```

MergeSort: Merge

- Assuming L and R are sorted already, merge the two to create a single sorted array

```
MERGE(L, R)
create array B of length length[L] + length[R]
i ← 1
j ← 1
for k ← 1 to length[B]
        if j > length[R] or (i ≤ length[L] and L[i] ≤ R[j])
                B[k] ← L[i]
                i ← i + 1
        else
                B[k] ← R[j]
                j ← j + 1
return B
```

Merge-Sort

- Running time?

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + D(n) + C(n) & \text{otherwise} \end{cases}$$

D(n): cost of splitting (dividing) the data

C(n): cost of merging/combining the data

Merge-Sort

- Running time?

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + D(n) + C(n) & \text{otherwise} \end{cases}$$

D(n): cost of splitting (dividing) the data - linear Θ(n)

C(n): cost of merging/combining the data – linear Θ(n)

Merge-Sort

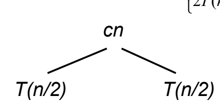
- Running time?

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + cn & \text{otherwise} \end{cases}$$

Which is?

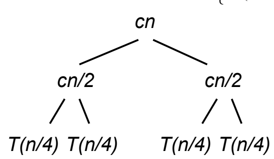
Merge-Sort

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + cn & \text{otherwise} \end{cases}$$



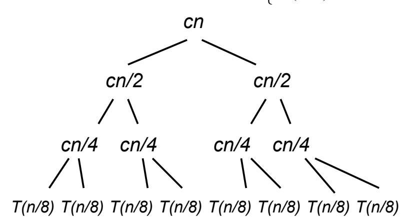
Merge-Sort

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + cn & \text{otherwise} \end{cases}$$



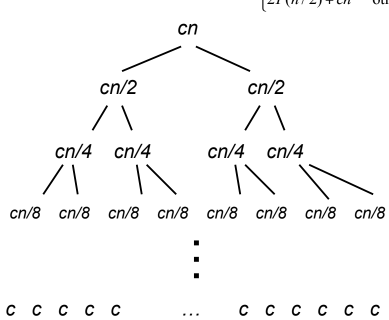
Merge-Sort

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + cn & \text{otherwise} \end{cases}$$



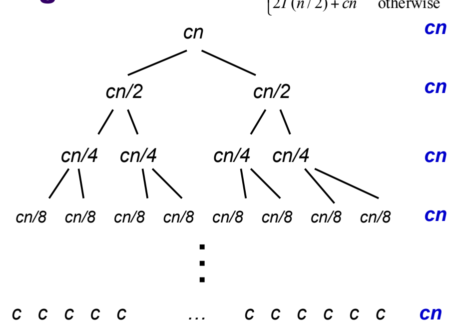
Merge-Sort

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + cn & \text{otherwise} \end{cases}$$



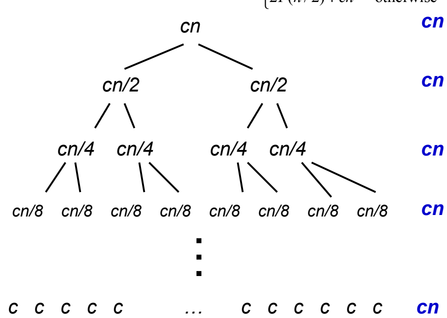
Merge-Sort

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + cn & \text{otherwise} \end{cases}$$



Merge-Sort

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + cn & \text{otherwise} \end{cases}$$



Depth?

Merge-Sort

- We can calculate the depth, by determining when the recursion gets to down to a small problem size, e.g. 1
- At each level, we divide by 2

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

$$2^d = n$$

$$\log 2^d = \log n$$

$$d \log 2 = \log n$$

$$d = \log_2 n$$

Merge-Sort

$$T(n) = \begin{cases} c & \text{if } n \text{ is small} \\ 2T(n/2) + cn & \text{otherwise} \end{cases}$$

- Running time?
  - Each level costs *cn*
  - log *n* levels
- *cn* log *n* = Θ(*n* log *n*)

Recurrence

- A function that is defined with respect to itself on smaller inputs

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

$$T(n) = 16T(n/4) + n$$

$$T(n) = 2T(n-1) + n^2$$

Why are we interested in recurrences?

- Computational cost of divide and conquer algorithms

$$T(n) = aT(n/b) + D(n) + C(n)$$

  - *a* subproblems of size *n/b*
  - *D(n)* the cost of dividing the data
  - *C(n)* the cost of recombining the subproblem solutions
- In general, the runtimes of most recursive algorithms can be expressed as recurrences

The challenge

- Recurrences are often easy to define because they mimic the structure of the program
- But… they do not directly express the computational cost, i.e. n, n^2, …
- We want to remove self-recurrence and find a more understandable form for the function

Three approaches

- **Substitution method**: when you have a good guess of the solution, prove that it's correct
- **Recursion-tree method**: If you don't have a good guess, the recursion tree can help. Then solve with substitution method.
- **Master method**: Provides solutions for recurrences of the form:

$$T(n) = aT(n/b) + f(n)$$

Substitution method

- Guess the form of the solution
- Then prove it's correct by induction

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

- Halves the input then constant amount of work

Guess?

Substitution method

- Guess the form of the solution
- Then prove it's correct by induction

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

- Halves the input then constant amount of work
- Similar to binary search:

Guess: $O(\log_2 n)$

Proof?

$$T(n) = T(n/2) + d = O(\log_2 n)?$$

Ideas?

Proof?

$$T(n) = T(n/2) + d = O(\log_2 n)?$$

Proof by induction!
- Assume it's true for smaller T(k)
- prove that it's then true for current T(n)

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

- Assume $T(k) = O(\log_2 k)$ for all $k < n$
- Show that $T(n) = O(\log_2 n)$
- From our assumption, $T(n/2) = O(\log_2 n)$:

$$O(g(n)) = \left\{ f(n): \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

- From the definition of O: $T(n/2) \le c \log_2(n/2)$

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

- To prove that $T(n) = O(\log_2 n)$ we need to identify the appropriate constants:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

i.e. some constant c such that $T(n) \le c \log_2 n$

$$\begin{aligned} T(n) &= T(n/2) + d \\ &\le c \log_2(n/2) + d \\ &\le c \log_2 n - c \log_2 2 + d \\ &\le c \log_2 n - c + d \quad \text{residual} \\ &\le c \log_2 n \end{aligned}$$

if $c \ge d$

Base case?

- For an inductive proof we need to show two things:
  - Assuming it's true for $k < n$ show it's true for n
  - *Show that it holds for some base case*
- What is the base case in our situation?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \text{ is small} \\ T(n/2) + d & \text{otherwise} \end{cases}$$

$$T(n) = T(n-1) + n$$

- Guess the solution?
  - At each iteration, does a linear amount of work (i.e. iterate over the data) and reduces the size by one at each step
  - $O(n^2)$
- Assume $T(k) = O(k^2)$ for all $k < n$
  - again, this implies that $T(n-1) \le c(n-1)^2$
- Show $T(n) = O(n^2)$, i.e. $T(n) \le cn^2$

$$\begin{aligned} T(n) &= T(n-1) + n \\ &\le c(n-1)^2 + n \\ &= c(n^2 - 2n + 1) + n \\ &= cn^2 - 2cn + c + n \quad \text{residual} \\ &\le cn^2 \end{aligned}$$

if

$$\begin{aligned} -2cn + c + n &\le 0 \\ -2cn + c &\le -n \\ c(-2n+1) &\le -n \\ c &\ge \frac{n}{2n-1} \\ c &\ge \frac{1}{2 - 1/n} \end{aligned}$$

which holds for any $c \ge 1$ for $n \ge 1$

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

- Guess the solution?
  - Recurses into 2 sub-problems that are half the size and performs some operation on all the elements
  - $O(n \log n)$
- What if we guess wrong, e.g. $O(n^2)$?
- Assume $T(k) = O(k^2)$ for all $k < n$
  - again, this implies that $T(n/2) \le c(n/2)^2$
- Show that $T(n) = O(n^2)$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\le 2c(n/2)^2 + n \\ &= 2cn^2/4 + n \\ &= 1/2cn^2 + n \\ &= cn^2 - (1/2cn^2 - n) \\ &\le cn^2 \end{aligned}$$

residual

if

$$-(1/2cn^2 - n) \le 0$$

$$-1/2cn^2 + n \le 0$$

$$cn \ge 2$$

overkill?

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

- What if we guess wrong, e.g. $O(n)$?
- Assume $T(k) = O(k)$ for all $k < n$
  - again, this implies that $T(n/2) \le c(n/2)$
- Show that $T(n) = O(n)$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\le 2cn/2 + n \\ &= cn + n \\ &\le cn \end{aligned}$$

factor of n so we can just roll it in?

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

- What if we guess wrong, e.g. $O(n)$?
- Assume $T(k) = O(k)$ for all $k < n$
  - again, this implies that $T(n/2) \le c(n/2)$
- Show that $T(n) = O(n)$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\le 2cn/2 + n \\ &= cn + n \\ &\le cn \end{aligned}$$

Must prove the exact form!

$cn+n \le cn$??

~~factor of n so we can just roll it in?~~

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

- Prove $T(n) = O(n \log_2 n)$
- Assume $T(k) = O(k \log_2 k)$ for all $k < n$
  - again, this implies that $T(k) = ck \log_2 k$
- Show that $T(n) = O(n \log_2 n)$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\le 2cn/2 \log(n/2) + n \\ &\le cn(\log_2 n - \log_2 2) + n \\ &\le cn \log_2 n - cn + n \quad \text{residual} \\ &\le cn \log_2 n \end{aligned}$$

if $cn \ge n$, $c > 1$

Changing variables

$$T(n) = 2T(\sqrt{n}) + \log n$$

- Guesses?
- We can do a variable change: let $m = \log_2 n$ (or $n = 2^m$)

$$T(2^m) = 2T(2^{m/2}) + m$$

- Now, let $S(m)=T(2^m)$

$$S(m) = 2S(m/2) + m$$

Changing variables

$$S(m) = 2T(m/2) + m$$

- Guess? $S(m) = O(m \log m)$

$$T(n) = T(2^m) = S(m) = O(m \log m)$$

substituting m=log n

$$T(n) = O(\log n \log \log n)$$

Recursion Tree

- Guessing the answer can be difficult

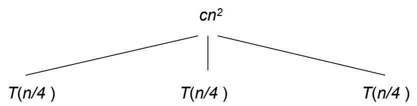
$$T(n) = 3T(n/4) + n^2$$

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

- The recursion tree approach
  - Draw out the cost of the tree at each level of recursion
  - Sum up the cost of the levels of the tree
    - Find the cost of each level with respect to the depth
    - Figure out the depth of the tree
    - Figure out (or bound) the number of leaves
  - Verify your answer using the substitution method

$$T(n) = 3T(n/4) + n^2$$

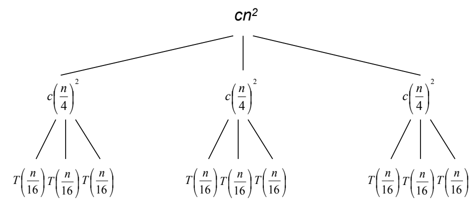
cost



cn^2

$$T(n) = 3T(n/4) + n^2$$

cost

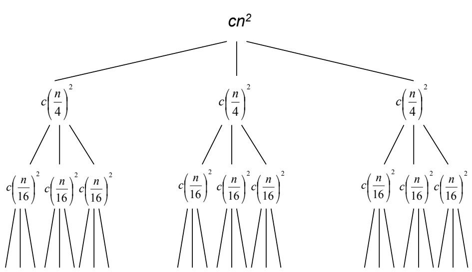


cn^2

$3/16cn^2$

$$T(n) = 3T(n/4) + n^2$$

cost



cn^2

$3/16cn^2$

$(3/16)^2cn^2$

What is the cost at each level? $\left(\frac{3}{16}\right)^d cn^2$

What is the depth of the tree?

- At each level, the size of the data is divided by 4

$$\frac{n}{4^d} = 1$$

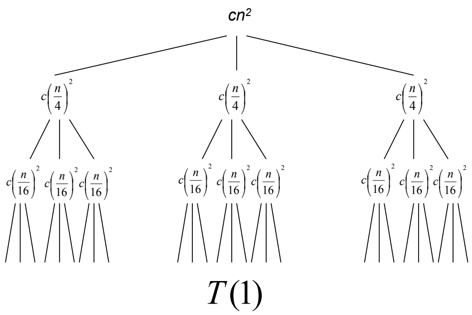
$$\log\left(\frac{n}{4^d}\right) = 0$$

$$\log n - \log 4^d = 0$$

$$d \log 4 = \log n$$

$$d = \log_4 n$$

$$T(n) = 3T(n/4) + n^2$$



How many leaves are there?

How many leaves?

- How many leaves are there in a complete ternary tree of depth *d*?

$$3^d = 3^{\log_4 n}$$

Total cost

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \ldots + \left(\frac{3}{16}\right)^{d-1} cn^2 + \Theta(3^{\log_4 n})$$

$$= cn^2 \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i + \Theta(3^{\log_4 n})$$

$$< cn^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + \Theta(3^{\log_4 n})$$

$$= \frac{1}{1-(3/16)} cn^2 + \Theta(3^{\log_4 n})$$

$$= \frac{16}{13} cn^2 + \Theta(3^{\log_4 n})$$?

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

let x = 3/16

Total cost $T(n) = \frac{16}{13}cn^2 + \Theta(3^{\log_4 n})$

$$3^{\log_4 n} = 4^{\log_4 3^{\log_4 n}}$$

$$= 4^{\log_4 n \log_4 3}$$

$$= 4^{\log_4 n^{\log_4 3}}$$

$$= n^{\log_4 3}$$

$$T(n) = \frac{16}{13}cn^2 + \Theta(n^{\log_4 3})$$

$$T(n) = O(n^2)$$

Verify solution using substitution

$$T(n) = 3T(n/4) + n^2$$

- Assume *T(k) = O(k²)* for all *k < n*
- Show that *T(n) = O(n²)*

- Given that *T(n/4) = O((n/4)²)*, then

$$O(g(n)) = \left\{ f(n): \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

- $T(n/4) \le c(n/4)^2$

$$T(n) = 3T(n/4) + n^2$$

- To prove that Show that $T(n) = O(n^2)$ we need to identify the appropriate constants:

$$O(g(n)) = \left\{ f(n): \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

i.e. some constant *c* such that $T(n) \le cn^2$

$$\begin{aligned} T(n) &= 3T(n/4) + n^2 \\ &\le 3c(n/4)^2 + n^2 \\ &= cn^2 3/16 + n^2 \\ &\le cn^2 \end{aligned}$$

if

$$c \ge \frac{16}{13}$$

Master Method

- Provides solutions to the recurrences of the form:

$$T(n) = aT(n/b) + f(n)$$

$$\text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for } \varepsilon > 0, \text{ then } T(n) = \Theta(n^{\log_b a})$$

$$\text{if } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \log n)$$

$$\text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for } \varepsilon > 0 \text{ and } af(n/b) \le cf(n) \text{ for } c < 1$$
$$\text{then } T(n) = \Theta(f(n))$$

$$T(n) = 16T(n/4) + n$$

$$\text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for } \varepsilon > 0, \text{ then } T(n) = \Theta(n^{\log_b a})$$
$$\text{if } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \log n)$$
$$\text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for } \varepsilon > 0 \text{ and } af(n/b) \le cf(n) \text{ for } c < 1$$
$$\text{then } T(n) = \Theta(f(n))$$

a = 16
b = 4
f(n) = *n*

$$n^{\log_b a} = n^{\log_4 16} = n^2$$

$$\text{is } n = O(n^{2-\varepsilon})?$$
$$\text{is } n = \Theta(n^2)?$$
$$\text{is } n = \Omega(n^{2+\varepsilon})?$$

Case 1: Θ(*n*²)

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

if $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$
then $T(n) = \Theta(f(n))$

a = 1
b = 2
f(n) = 2^n

$n^{\log_b a} = n^{\log_2 1} = n^0$

is $2^n = O(n^{0-\varepsilon})$?
is $2^n = \Theta(n^0)$?
is $2^n = \Omega(n^{0+\varepsilon})$?

Case 3?

is $2^{n/2} \le c2^n$ for $c < 1$?

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

if $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$
then $T(n) = \Theta(f(n))$

is $2^{n/2} \le c2^n$ for $c < 1$?

Let c = 1/2

$2^{n/2} \le (1/2)2^n$
$2^{n/2} \le 2^{-1}2^n$
$2^{n/2} \le 2^{n-1}$

T(*n*) = Θ(2*ⁿ*)

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

if $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$
then $T(n) = \Theta(f(n))$

a = 2
b = 2
f(n) = *n*

$n^{\log_b a} = n^{\log_2 2} = n^1$

is $n = O(n^{1-\varepsilon})$?
is $n = \Theta(n^1)$?
is $n = \Omega(n^{1+\varepsilon})$?

Case 2: Θ(*n* log *n*)

$$T(n) = 16T(n/4) + n!$$

if $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$
then $T(n) = \Theta(f(n))$

a = 16
b = 4
f(n) = *n!*

$n^{\log_b a} = n^{\log_4 16} = n^2$

is $n! = O(n^{2-\varepsilon})$?
is $n! = \Theta(n^2)$?
is $n! = \Omega(n^{2+\varepsilon})$?

Case 3?

is $16(n/4)! \le cn!$ for $c < 1$?

$$T(n) = 16T(n/4) + n!$$

if $f(n) = O(n^{\log_b a-\varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a}\log n)$
if $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$
then $T(n) = \Theta(f(n))$

is $16(n/4)! \le cn!$ for $c < 1$?

Let c = 1/2

$cn! = 1/2n!$

$> (n/2)!$

therefore,

$16(n/4)! \le (n/2)! < 1/2n!$

$T(n) = \Theta(n!)$

$$T(n) = \sqrt{2}T(n/2) + \log n$$

if $f(n) = O(n^{\log_b a-\varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a}\log n)$
if $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$
then $T(n) = \Theta(f(n))$

a = $\sqrt{2}$
b = 2
f(n) = log*n*

$n^{\log_b a} = n^{\log_2 \sqrt{2}} = n^{\log_2 2^{1/2}} = \sqrt{n}$

is $\log n = O(n^{1/2-\varepsilon})$?
is $\log n = \Theta(n^{1/2})$?
is $\log n = \Omega(n^{1/2+\varepsilon})$?

Case 1: $\Theta(\sqrt{n})$

$$T(n) = 4T(n/2) + n$$

if $f(n) = O(n^{\log_b a-\varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a}\log n)$
if $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$
then $T(n) = \Theta(f(n))$

a = 4
b = 2
f(n) = *n*

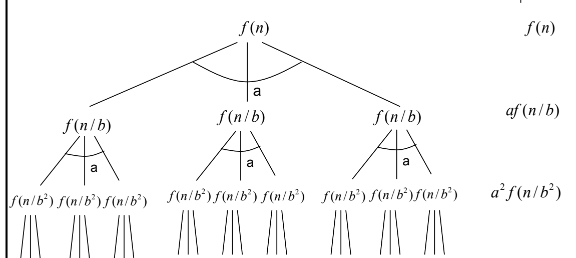
$n^{\log_b a} = n^{\log_2 4} = n^2$

is $n = O(n^{2-\varepsilon})$?
is $n = \Theta(n^2)$?
is $n = \Omega(n^{2+\varepsilon})$?

Case 1: $\Theta(n^2)$

Why does the master method work?

$$T(n) = aT(n/b) + f(n)$$



What is the depth of the tree?

- At each level, the size of the data is divided by b

$$\frac{n}{b^d} = 1$$

$$\log\left(\frac{n}{b^d}\right) = 0$$

$$\log n - \log 4^b = 0$$

$$d \log b = \log n$$

$$d = \log_b n$$

How many leaves?

- How many leaves are there in a complete *a*-ary tree of depth *d*?

$$a^d = a^{\log_b n}$$

$$= n^{\log_b a}$$

Total cost

if $f(n) = O(n^{\log_b a-\varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

if $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$ then $T(n) = \Theta(f(n))$

$$T(n) = cf(n) + af(n/b) + a^2 f(n/b^2) + \ldots + a^{n-1} f(n/b^{n-1}) + \Theta(n^{\log_b a3})$$

$$= \sum_{i=0}^{\log_b n-1} a^i f(n/b^i) + \Theta(n^{\log_b a})$$

Case 1: cost is dominated by the cost of the leaves

$$= \sum_{i=0}^{\log_b n-1} a^i f(n/b^i) < \Theta(n^{\log_b a})$$

Total cost

if $f(n) = O(n^{\log_b a-\varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

if $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$ then $T(n) = \Theta(f(n))$

$$T(n) = cf(n) + af(n/b) + a^2 f(n/b^2) + \ldots + a^{n-1} f(n/b^{n-1}) + \Theta(n^{\log_b a3})$$

$$= \sum_{i=0}^{\log_b n-1} a^i f(n/b^i) + \Theta(n^{\log_b a})$$

Case 2: cost is evenly distributed across tree

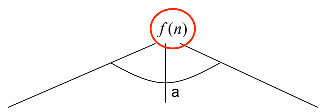
As we saw with mergesort, log *n* levels to the tree and at each level *f(n)* work

Total cost

$$\text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for } \varepsilon > 0, \text{ then } T(n) = \Theta(n^{\log_b a})$$
$$\text{if } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \log n)$$
$$\text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for } \varepsilon > 0 \text{ and } af(n/b) \le cf(n) \text{ for } c < 1$$
$$\text{then } T(n) = \Theta(f(n))$$

$$T(n) = cf(n) + af(n/b) + a^2 f(n/b^2) + ... + a^{n-1} f(n/b^{n-1}) + \Theta(n^{\log_b a3})$$
$$= \sum_{i=0}^{\log_b n - 1} a^i f(n/b^i) + \Theta(n^{\log_b a})$$

Case 3: cost is dominated by the cost of the root



Don't shoot the messenger

- Why do we care about substitution method and recurrence tree method? Master method is much easier.

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

- Some recurrences don't fit the mold!

Other forms of the master method

$$T(n) = aT(n/b) + O(n^d)$$

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Recurrences

$$T(n) = 2T(n/3) + d \qquad T(n) = 7T(n/7) + n$$

$$\text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for } \varepsilon > 0, \text{ then } T(n) = \Theta(n^{\log_b a})$$
$$\text{if } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \log n)$$
$$\text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for } \varepsilon > 0 \text{ and } af(n/b) \le cf(n) \text{ for } c < 1$$
$$\text{then } T(n) = \Theta(f(n))$$

$$T(n) = T(n-1) + \log n \qquad T(n) = 8T(n/2) + n^3$$