CS583 Lecture 02

Jana Kosecka

some materials here are based on E. Demaine , D. Luebke slides

Previously

- Sample algorithms
- Exact running time, pseudo-code
- Approximate running time
- Worst case analysis
- Best case analysis

Rules of thumb

- Multiplicative constants can be omitted
- n^a dominates n^b if $a > b$; e.g. n^2 dominates n
- Any exponential dominates any polynomial
- E.g. 3^n dominates n^5
- Any polynomial dominates any logarithm
- E.g. n dominates $(\log n)^3$

Today’s topics

- Solving recurrences
- Substitution method
- Iteration methods
- Recursion tree
- Masters’s theorem

Recurrence

- Methods for solving recurrences
- Some examples last time
- Expanding the reccurrence
- Recursion tree
- Technical issues; assume that $n = 2^k$

Solving Recurrences

- Another option is “iteration method”
  - Expand the recurrence
  - Work some algebra to express as a summation
  - Evaluate the summation
- We will show several examples

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

- s(n) = c + s(n-1)

  = c + c + s(n-2) = 2c + s(n-2)

  = 2c + c + s(n-3) = 3c + s(n-3) = …

  = kc + s(n-k) = ck + s(n-k)

- So far for n >= k we have

  s(n) = ck + s(n-k)

- What if k = n?

  s(n) = cn + s(0) = cn

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

- Thus in general s(n) = cn

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- s(n)

= n + s(n-1)

= n + n-1 + s(n-2)

= n + n-1 + n-2 + s(n-3)

= n + n-1 + n-2 + n-3 + s(n-4)

= …

= n + n-1 + n-2 + n-3 + … + n-(k-1) + s(n-k)

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- s(n)

= n + s(n-1)

= n + n-1 + s(n-2)

= n + n-1 + n-2 + s(n-3)

= n + n-1 + n-2 + n-3 + s(n-4)

= …

= n + n-1 + n-2 + n-3 + … + n-(k-1) + s(n-k)

$$= \sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

- What if k = n?

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

- What if k = n?

$$\sum_{i=1}^{n} i \quad + \quad s(0) \quad = \quad \sum_{i=1}^{n} i \quad + \quad 0 \quad = \quad n\frac{n+1}{2}$$

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

- What if k = n?

$$\sum_{i=1}^{n} i \quad + \quad s(0) \quad = \quad \sum_{i=1}^{n} i \quad + \quad 0 \quad = \quad n\frac{n+1}{2}$$

- Thus in general

$$s(n) \quad = \quad n\frac{n+1}{2}$$

$$T(n) = \begin{cases} c & n = 1 \\ 2T(n/2) + c & n > 1 \end{cases}$$

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

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

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

  $= 2^3T(n/2^3) + 7c$

  $= 2^3(2T(n/2^3/2) + c) + 7c = 2^4T(n/2^4) + 15c$

  ….

  $= 2^kT(n/2^k) + (2^k - 1)c$

$$T(n) = \begin{cases} c & n = 1 \\ 2T(n/2) + c & n > 1 \end{cases}$$

- So far we have
  - $T(n) = 2^k T(n/2^k) + (2^k - 1)c$
- What if $k = \lg n$?
  - $T(n) = 2^{\lg n}\, T(n/2^{\lg n}) + (2^{\lg n} - 1)c$

    $= n\, T(n/n) + (n - 1)c$

    $= n\, T(1) + (n-1)c$

    $= nc + (n-1)c = (2n - 1)c$

Bounding Functions

- non-recursive algorithms
  - set up a sum for the number of times the basic operation is executed
  - simplify the sum and determine the order of growth (using asymptotic notation)

1. $\sum_{1=1}^{n} 1 = 1 + 1 + \cdots + 1 = n \in \Theta(n)$

2. $\sum_{1=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{n^2}{2} \in \Theta(n^2)$

3. $\sum_{1=1}^{n} i^2 = 1 + 4 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{n^3}{3} \in \Theta(n^3)$

4. $\sum_{1=0}^{n} a^i = 1 + a^1 + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1}, \forall a \neq 1, \in \Theta(a^n)$

5. $\sum a_i + b_i = \sum a_i + \sum b_i$

6. $\sum c a_i = c \sum a_i$

7. $\sum_{1=0}^{n} a_i = \sum_{1=0}^{m} a_i + \sum_{1=m+1}^{n} a_i$

Substitution Method

- Most general method for solving recurrences
- Guess the form of solution
- Verify by induction
- Solve for constants

- Induction method of mathematical proof to establish a fact for all natural numbers

Induction Review

- Show the fact holds for base case, e.g. P(0) is true
- Form inductive hypothesis: Show that if P(k) holds then it also holds for P(k+1) => this implies that P(n) holds
- Example: Show that

$$0 + 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

Example

- Example $T(n) = 4T(n/4) + 4$
- Assume that $T(1) = \Theta(1)$
- Guess $O(n^3)$
- Assume that $T(k) \leq ck^3$ for $k < n$
- Prove $T(n) \leq cn^3$ by induction

Example of substitution

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

$$\leq 4c(n/2)^3 + n$$

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

$$= cn^3 - ((c/2)n^3 - n)$$ ← desired - residual

$$\leq cn^3$$ ← desired

- Whenever $(c/2)n^3 - n \geq 0$ for example
- If $c \geq 2; n \geq 1$

Example cont

- Handle initial conditions, to ground the induction with the base case
- Base case $T(1) = \Theta(1)$ for all $n < n_0$
- For $1 \leq n \leq, n_0$ we have $\Theta(1) \leq cn^3$

if we pick c big enough

This bound is not tight !

Tighter upper bound

- Prove that $T(n) = O(n^2)$

$$T(n) = 4T(n/2) + n$$
$$\leq 4c(n/2)^2 + n$$
$$= cn^2 + n$$
$$\leq O(n^2)$$ Wrong !must prove inductive hyp.
$$= cn^2 - (-n)$$
$$\leq cn^2$$ For no choice of constant

Tighter upper bound

- Strengthen induction hypothesis $T(k) \le c_1 k^2 - c_2 k$

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4(c_1(n/2)^2 - c_2(n/2)) + n \\
&= c_1 n^2 - 2c_2 n + n \\
&= c_1 n^2 - c_2 n - (c_2 n - n) \\
&\le c_1 n^2 - c_2 n
\end{aligned}
$$

Substitution

- we can also guess that

  $T(n) = 2T(\frac{n}{2}) + n \in O(n)$, where $T(1) = 1$.

- Another strategy: change of variables

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

Recursion Tree

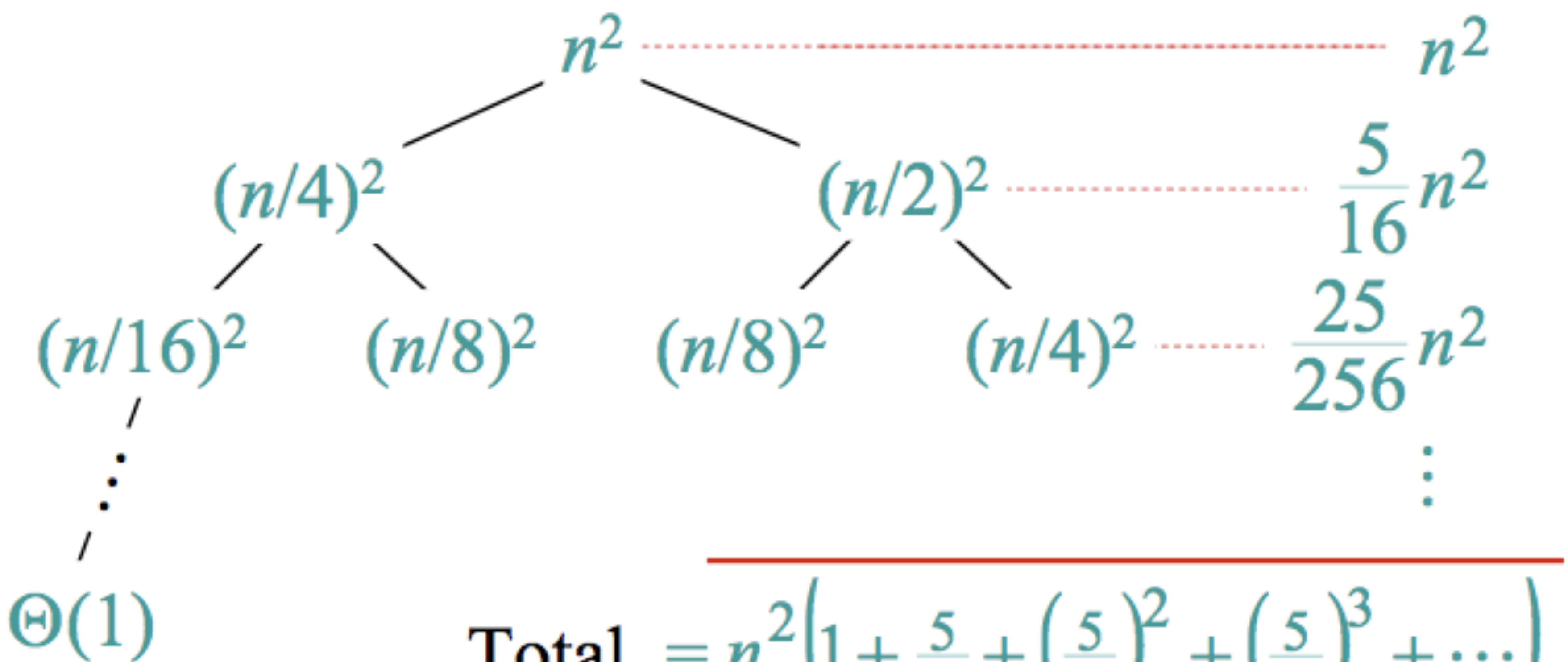
- Recursion tree is good for make an initial guess of the bound
- Build a recursion tree for $T(n) = 2T(n/2) + cn$

Recursion Tree Example

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

Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



$$\text{Total} = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$$

$$= \Theta(n^2)$$ *geometric series*

Masters Method

- Cookbook method for solving recurrences of the type

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

Master Theorem

- If $T(n) = aT(n/b) + f(n)$
- Idea compare the rate of growth of $f(n)$ with $n^{\log_b a}$
- $f(n)$ grows polynomialy slower then $n^{\log_b a}$
- Solution is $T(n) = \Theta(n^{\log_b a})$

CASE 1: $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a})$.

Masters Theorem

- Idea compare the rate of growth of $f(n)$ with $n^{\log_b a}$
- $f(n)$ grows at similar rate then $n^{\log_b a}$
- Solution is $T(n) = \Theta(n^{\log_b a} \lg n)$

Master Theorem

- If $T(n) = aT(n/b) + f(n)$
- Idea compare the rate of growth of $f(n)$ with $n^{\log_b a}$
- $f(n)$ grows polynomialy faster then $n^{\log_b a}$
- Solution is $T(n) = \Theta(f(n))$

Case 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$, and regularity condition
$\Rightarrow T(n) = \Theta(f(n))$.

- Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$

Master Theorem

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

CASE 1: $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a})$.

CASE 2: $f(n) = \Theta(n^{\log_b a} \lg^{[illegible]} n)$, constant [illegible]

$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{[illegible]} n)$.

CASE 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$,
and regularity condition

$\Rightarrow T(n) = \Theta(f(n))$.

Case 1: $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a})$.

Case 2: $f(n) = \Theta(n^{\log_b a} \lg n)$, constant

$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg \quad n)$.

Case 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$, and regularity condition

$\Rightarrow T(n) = \Theta(f(n))$.

- Merge Sort Example
- CASE 2

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

$$a = 2, b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n$$

$$k = 0 \Rightarrow T(n) = \Theta(n \lg n)$$

Examples

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

Examples

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

Examples

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

Asymptotic Bounds for Some Common Functions

- Polynomials. $a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$.

Polynomial time. Running time is $O(n^d)$ for some constant d independent of the input size n.

- Logarithms. $O(\log_a n) = O(\log_b n)$ for any constants a, b > 0.

can avoid specifying the base

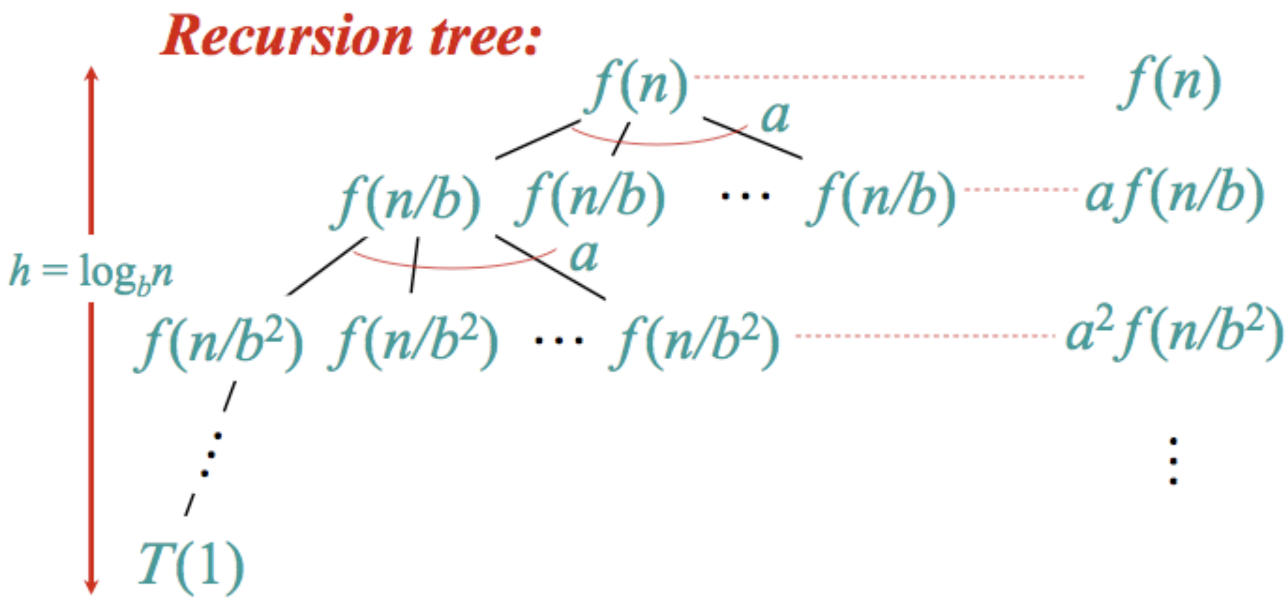
- Logarithms. For every x > 0, $\log n = O(n^x)$.

log grows slower than every polynomial

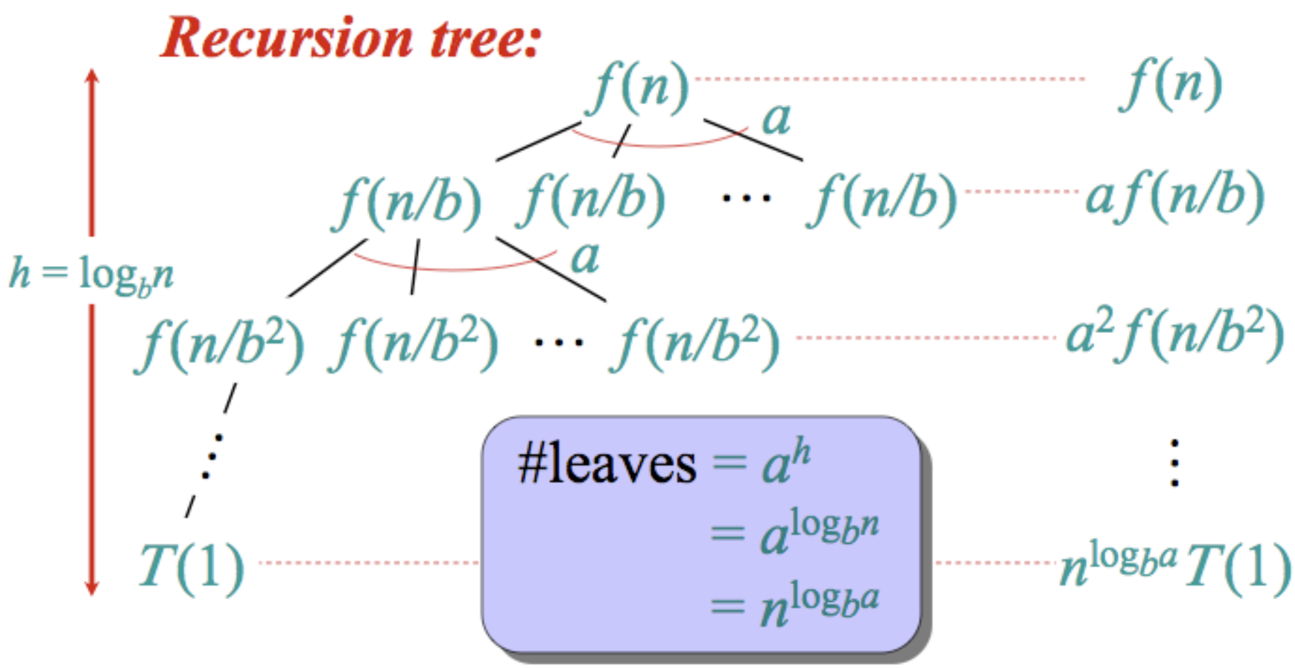
- Exponentials. For every r > 1 and every d > 0, $n^d = O(r^n)$.

every exponential grows faster than every polynomial

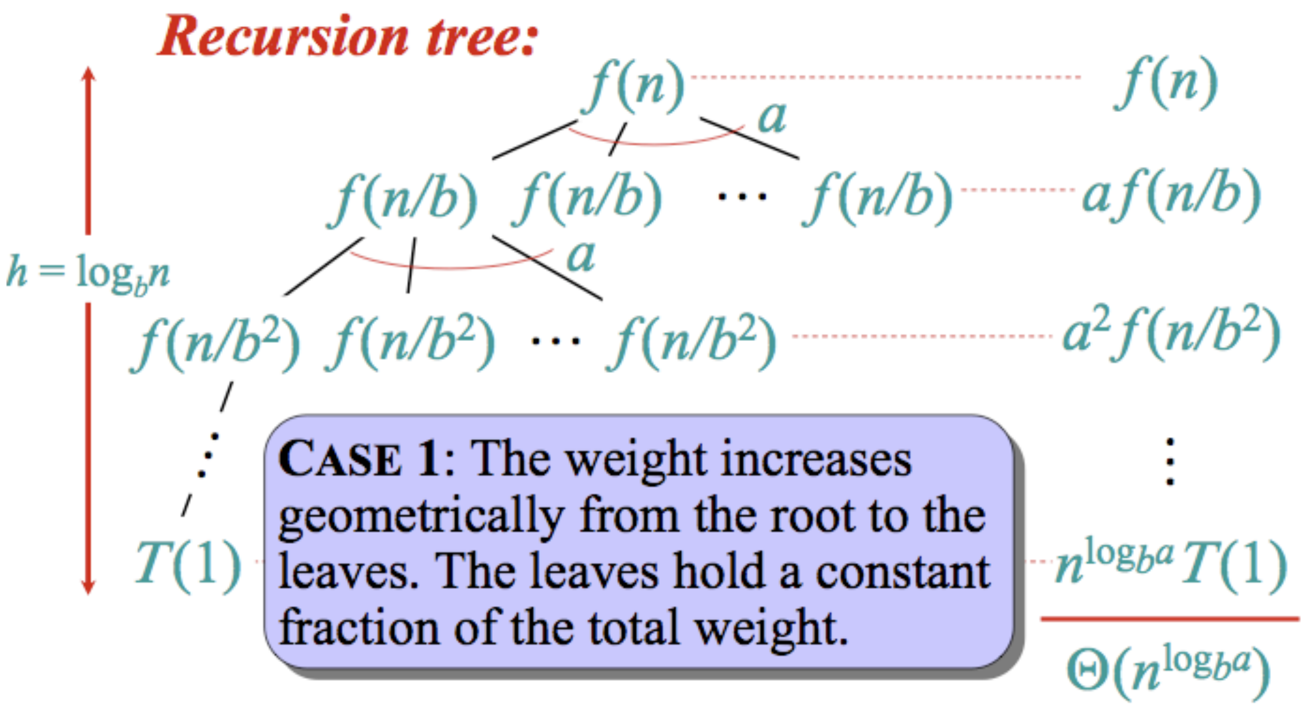
Masters Theorem via recursion tree



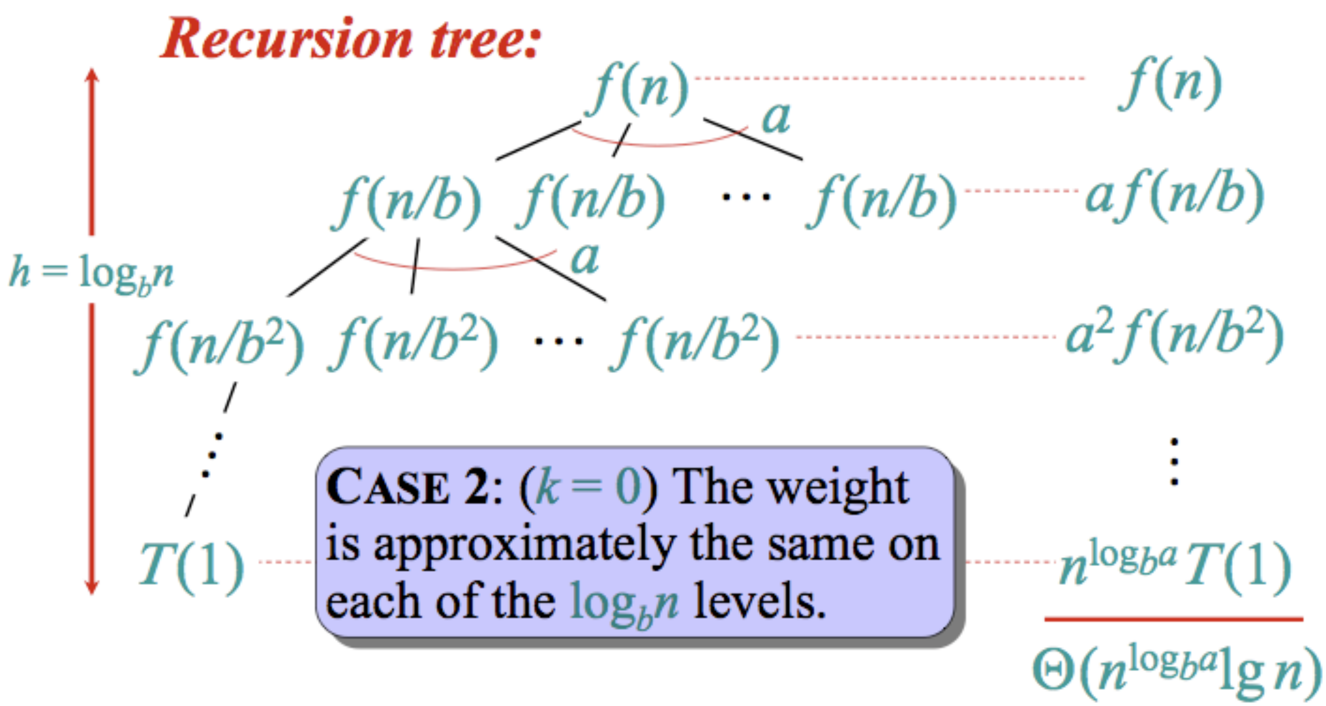
Masters Theorem via recursion tree



Masters Theorem via recursion tree

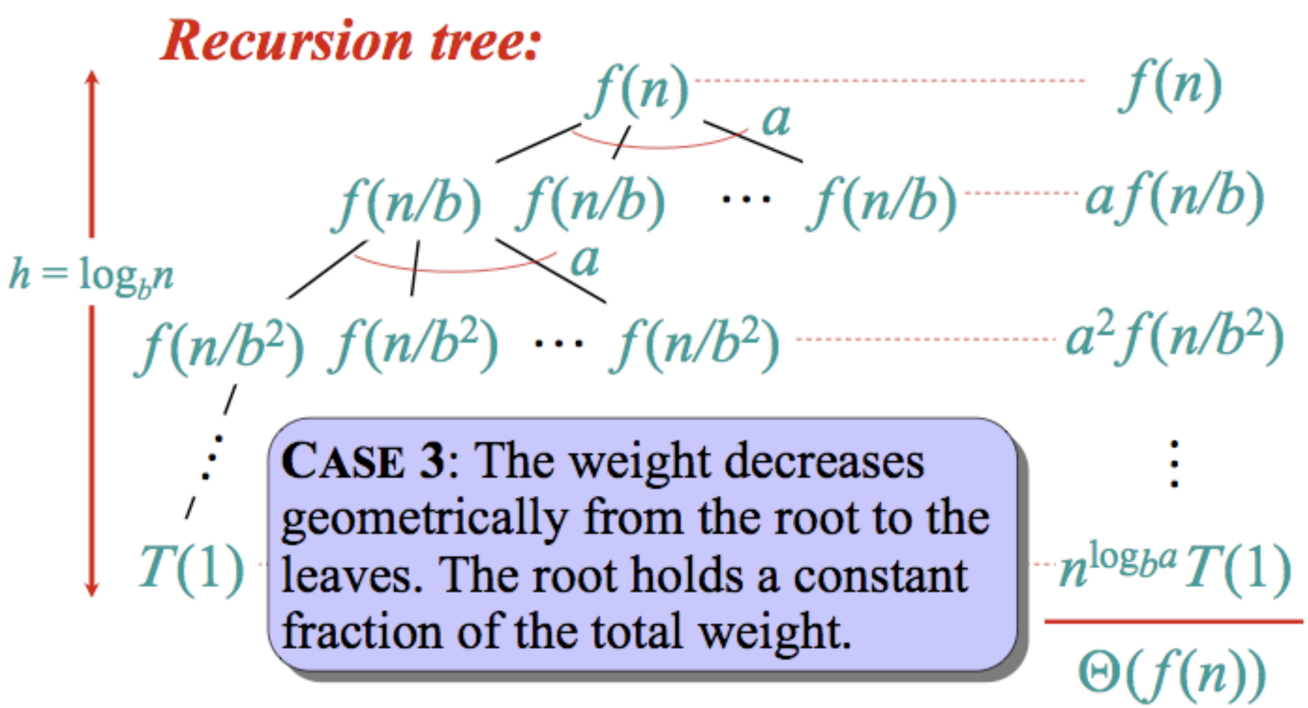


Masters Theorem via recursion tree



CASE 2: ($k = 0$) The weight is approximately the same on each of the $\log_b n$ levels.

Masters Theorem via recursion tree



Binary Search

- Find an element in the sorted array
- Divide and conquer algorithm

1. Divide: Check the middle element
2. Conquer: Recursively search one subarray
3. Combine: Trivial

Binary Search

- Find 9 in sorted array

3 5 7 8 9 12 15

Binary Search

- Recurrence equation

$$T(n) = 1T(n/2) + \Theta(1)$$

# of subproblems work dividing and combining

subproblem size

Binary Search

- Recurrence equation

$$T(n) = \underbrace{1}_{\text{\# of subproblems}} T(\underbrace{n/2}_{\text{subproblem size}}) + \underbrace{\Theta(1)}_{\text{work dividing and combining}}$$

- Analysis

Fibonacci Numbers

- Recursive definition

$$F_n = \begin{cases} 0 & if \quad n = 0; \\ 1 & if \quad n - 1; \\ F_{n-1} + F_{n-2} & if \quad n \geq 2 \end{cases}$$

0 1 1 2 3 5 8 13 21 34

Probabilistic Analysis

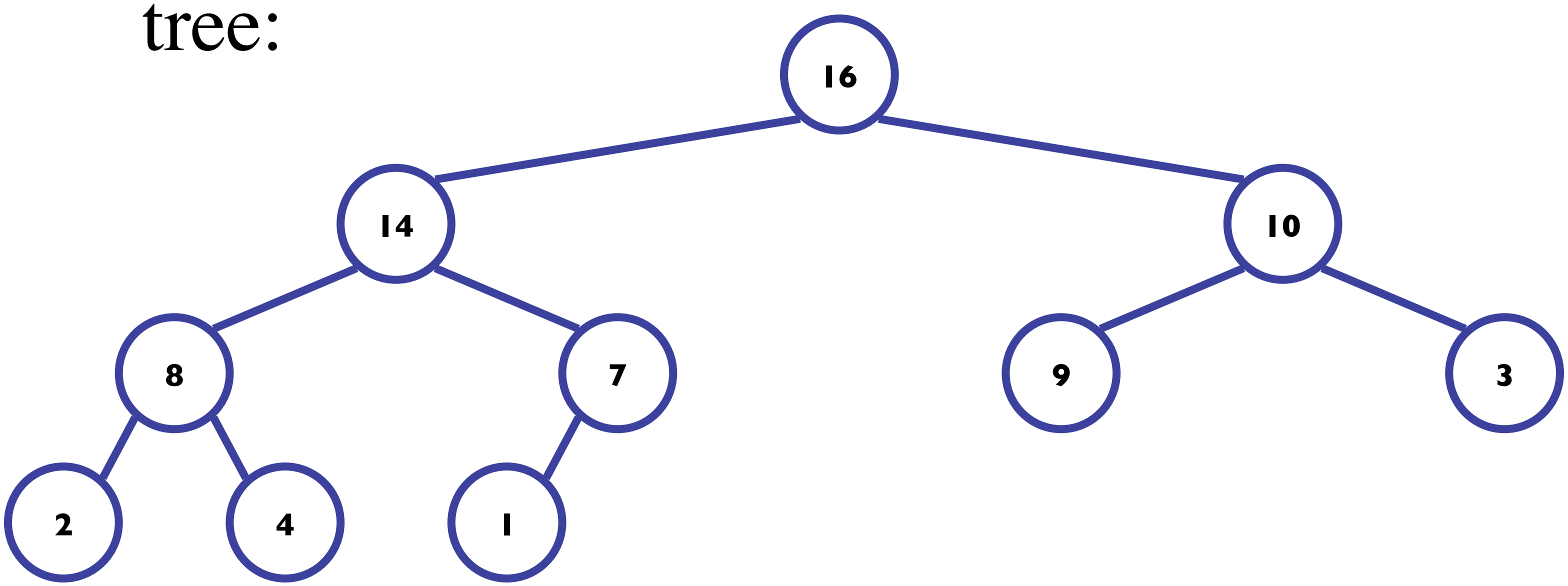
- use of probability theory in the analysis of algorithms
- To perform a probabilistic analysis, we have to **make assumptions on the distribution** of inputs
- After such assumption, we compute an **expected running time** that is computed over the distribution of all possible inputs
- We will return to it later

Sorting Continued

- So far we've talked about two algorithms to sort an array of numbers
  - What is the advantage of merge sort?
  - What is the advantage of insertion sort?
- Next on the agenda: *Heapsort*
  - Combines advantages of both previous algorithms

Heaps

- A *heap* can be seen as a complete binary tree:

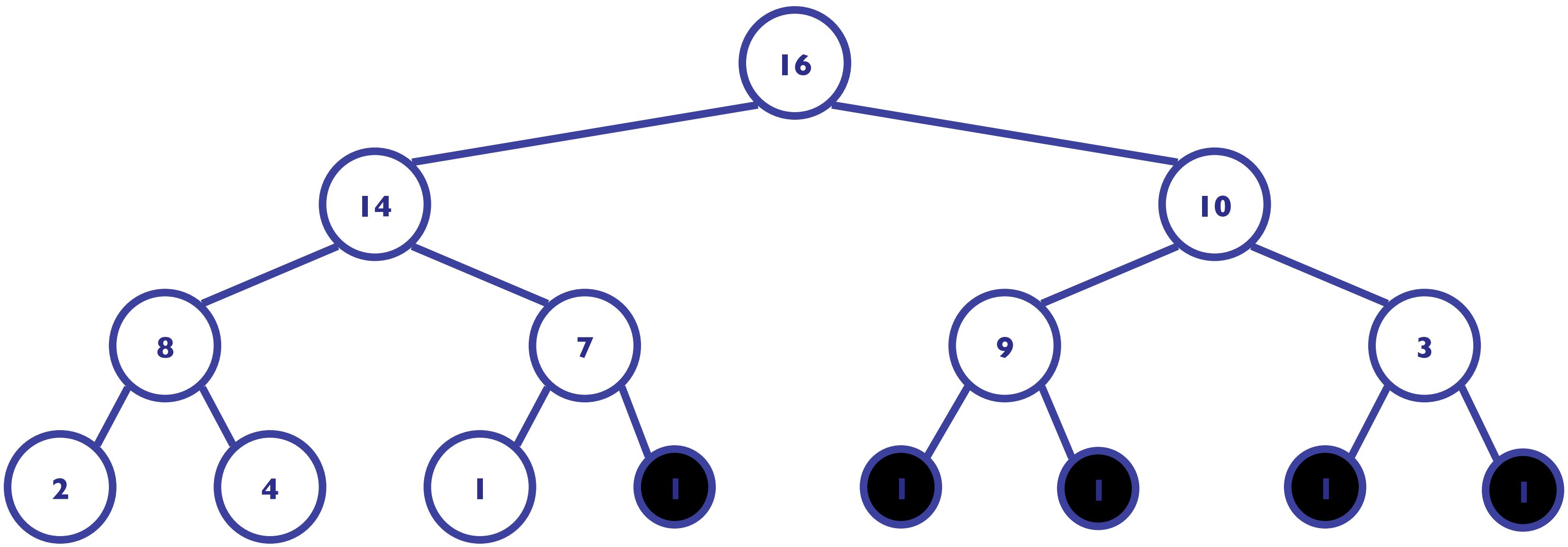


What makes a binary tree complete?

Is the example above complete?

Heaps

- A *heap* can be seen as a complete binary tree:

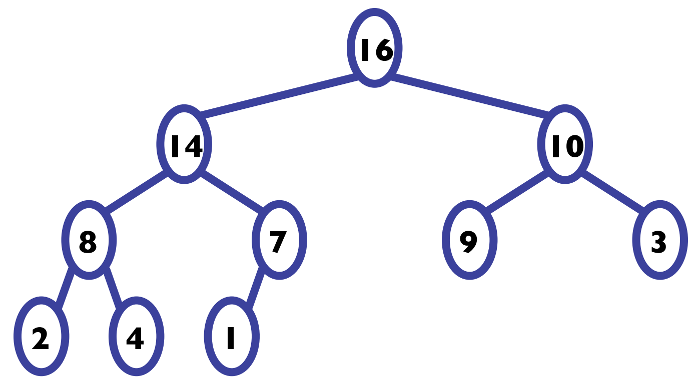


The book calls them "nearly complete" binary trees; can think of unfilled slots as null pointers

Heaps

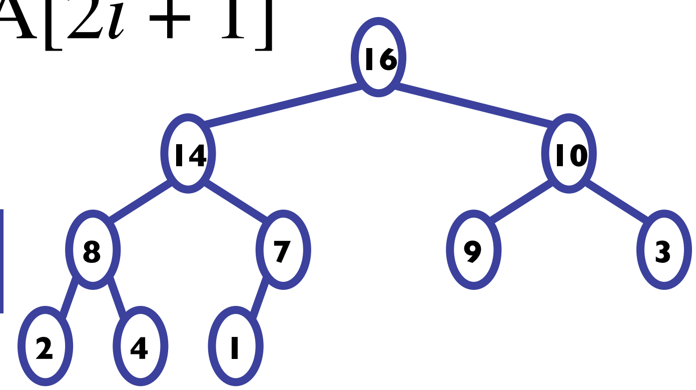
- In practice, heaps are usually implemented as arrays:

A =	16	14	10	8	7	9	3	2	4	1	=



Heaps

- To represent a complete binary tree as an array:
  - The root node is A[1]
  - Node *i* is A[*i*]
  - The parent of node *i* is A[*i*/2] (note: integer divide)
  - The left child of node *i* is A[2*i*]
  - The right child of node *i* is A[2*i* + 1]



A =	16	14	10	8	7	9	3	2	4	1	=

Referencing Heap Elements

- So…

```
Parent(i) { return ⌊i/2⌋; }
Left(i) { return 2*i; }
right(i) { return 2*i + 1; }
```

- An aside: *How would you implement this most efficiently?*
- Another aside: *Really?*

The Heap Property

- Heaps also satisfy the *heap property*:

  $A[Parent(i)] \geq A[i]$ for all nodes $i > 1$

  - In other words, the value of a node is at most the value of its parent
  - *Where is the largest element in a heap stored?*
- Definitions:
  - The *height* of a node in the tree = the number of edges on the longest downward path to a leaf
  - The height of a tree = the height of its root

Heap Height

- *What is the height of an n-element heap? Why?*
- This is nice: basic heap operations take at most time proportional to the height of the heap