Lecture 2: February 8, 2001

Administrative stuff:

- Surveys due today.
- PS 1 Out.
- Online handouts: Notes for L1.

Today:

- Asymptotic notation: O, Ω, Θ
  - Definition
  - Usage
  - Properties
- Solving recurrences
  - Iteration method
  - Substitution method
  - Master method

Asymptotic notation

- O-notation (upper bounds)
  - $f(n) = O(g(n))$ if $\exists$ const c, n_0 such that $0 \leq f(n) \leq cg(n), \forall n \geq n_0$.
  - e.g. $2n^2 = O(n^3)$ $(c = 1, n_0 = 2)$
    - $=$ is funny "one-way" equality
    - n^2, n^3 are functions, not values, but writing carefully is too tedious.

    Thus O-notation is sloppy, but convenient.

Though being sloppy, must understand what O-notation *really* means.

Think of $O(g(n))$ as a *set* of functions:

$O(g(n)) = \{f(n) : \exists$ const c, n_0 such that $0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$

Then $2n^2 \in O(n^3)$

Example

$f(n) = n^2 + O(n)$ means
$f(n) = n^2 + h(n)$ for some $h(n) \in O(n)$.

Note:
O-notation is an *upper-bound* notation.
Makes no sense to say $f(n)$ is at least $O(n)$.
Why?

Ω-notation (lower bounds)

$\Omega(g(n)) = \{f(n) : \exists c, n_0$ such that $0 \leq cg(n) \leq f(n), \forall n \geq n_0\}$.

e.g. $\sqrt{n} = \Omega(\lg n)$ $(c = 1, n_0 = 16)$

Θ-notation (tight bounds)

$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}$.

e.g.

$$\frac{1}{2}n^2 + 2n = \Theta(n^2)$$

$$0 \leq \frac{1}{4}n^2 \leq \frac{1}{2}n^2 + 2n \leq \frac{3}{4}n^2$$

$$0 \leq \frac{1}{4}n \leq \frac{1}{2}n + 2 \leq \frac{3}{4}n$$

$$0 \leq n \leq 2n + 8 \leq 3n$$

$$2n + 8 \leq 3n$$

$$8 \leq n (c_1 = 1/4, c_2 = 3/4, n_0 = 8)$$

Theorem 1: Leading constants and low-order terms don't matter. (As just seen in example.)

Justification (not proof, just handwaving): Can choose constant big enough to make high-order term swamp other terms.

Theorem 2: (O and Ω) $\Leftrightarrow \Theta$.
(Left as [really easy] exercise.)

Solving Recurrences

Like solving integrals, differential equations, etc. (learn a few tricks!)

I. Substitution Method
(most general)

1. **Guess** form of solution
2. **Verify** by induction
3. **Solve** for constants

Example of substitution method proof:

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

1. Guess that $T \in O(n^3)$, i.e., that T of form cn^3
2. Assume $T(k) \le ck^3$ for $k < n$ and
3. Prove $T(n) \le cn^3$ by induction
   (important: same constant!)

$$\begin{aligned} T(n) &= 4T(n/2) + n && \text{(recurrence)} \\ &\le 4c(n/2)^3 + n && \text{(ind. hypoth.)} \\ &= \frac{c}{2}n^3 + n && \text{(simplify)} \\ &= cn^3 - \left(\frac{c}{2}n^3 - n\right) && \text{(rearrange)} \\ &\le cn^3 \text{ if } c \ge 2 \text{ and } n \ge 1 && \text{(satisfy)} \end{aligned}$$

Thus $T(n) = O(n^3)$!

Initial conditions: Pick c big enough to handle $T(n) = \Theta(1)$ for $n < n_0$ for some n_0.

Note:

1. in last equality:
   To prove inductive step, try to write expression as

   $$\langle \text{answer you want} \rangle - \langle \text{ something } > 0 \rangle$$

2. did not show that $T(n) = \Omega(n)$ or $\Theta(n)$, thus n^3 is not a tight bound!
   Can show $O(n^2)$.

Try to show $T(n) = O(n^2)$

Assume $T(k) \le ck^2$

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4c(n/2)^2 + n \\ &= cn^2 + n \\ &= O(n^2) \\ &\le cn^2 \text{ for } no \text{ choice of } c > 0. \text{ Lose.} \end{aligned}$$

What went wrong?

Fallacious argument

Assume $T(k) \le ck^2$

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4c(n/2)^2 + n \\ &= cn^2 + n \\ &= O(n^2) \text{ — WRONG!} \\ &\le cn^2 \text{ for } no \text{ choice of } c > 0. \text{ Lose.} \end{aligned}$$

Didn't show same as inductive assertion!

Correct Proof

Idea: *Strengthen* inductive hypothesis by *subtracting lower-order term*:

Assume $T(k) \le c_1 k^2 - c_2 k$ for $k < n$

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4(c_1(n/2)^2 - c_2(n/2)) + n \\ &= c_1 n^2 - 2c_2 n + n \\ &= c_1 n^2 - c_2 n - (c_2 n - n) \\ &\le c_1 n^2 - c_2 n \text{ if } c_2 \ge 1 \end{aligned}$$

Pick c_1 big enough to handle initial conditions.

Also:
Can use substitution method for lower (Ω) bounds.

II. Iterating recurrences

(convert to summation)

Example

$$\begin{aligned} T(n) &= n + 4T(n/2) \\ &= n + 4(n/2 + 4T(n/4)) \\ &= n + 4(n/2 + 4(n/4 + 4T(n/8))) \\ &= n + 4(n/2 + 4(n/4 + 4(n/8 + 4T(n/16)))) \\ &= n + 2n + 4n + \cdots + 2^{\lg n} T(1) \\ &\qquad\qquad 2^{\lg n} = n \\ &= n \left(\sum_{k=0}^{\lg n - 1} 2^k \right) + \Theta(n) \end{aligned}$$

II. Iterating recurrences (cont.)

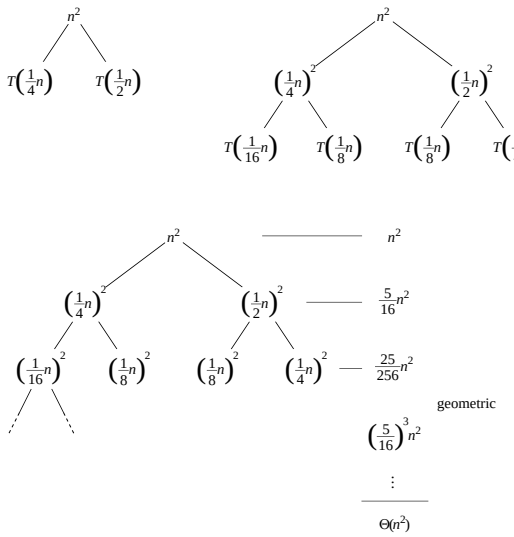
$$\begin{aligned} &= n \left(\sum_{k=0}^{\lg n - 1} 2^k \right) + \Theta(n) \\ &\qquad \text{Look up: } \sum_{k=0}^{t} x^k = \frac{x^{t+1} - 1}{x - 1} \text{ for } x \ne 1 \\ &= n \left(\frac{2^{\lg n} - 1}{2 - 1} \right) + \Theta(n) \\ &= \Theta(n^2) + \Theta(n) \\ &= \Theta(n^2) \end{aligned}$$

To iterate recurrences

- Should know rules and have intuition for arithmetic and geometric series (like series in the example).
- Math can be messy and hard. Often, use iterating recurrence to generate guess for substitution method.

Recursion tree: visualizing iteration

Construction of recursion tree
for $T(n) = T(n/4) + T(n/2) + n^2$:

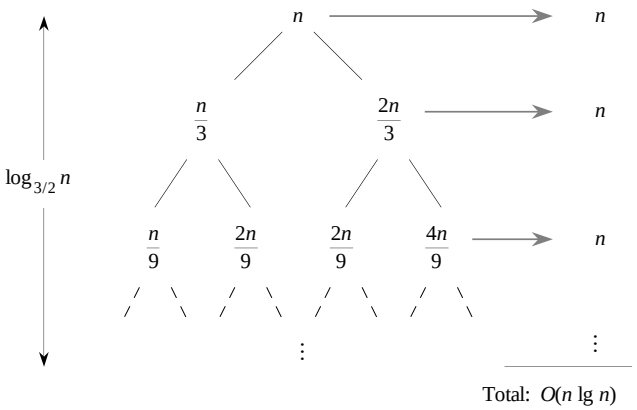


$$T(n) = n^2(1 + \frac{5}{16} + (\frac{5}{16})^2 + \cdots) = \frac{16}{11}n^2 = \Theta(n^2)$$

Another recurrence:

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

Recursion tree:



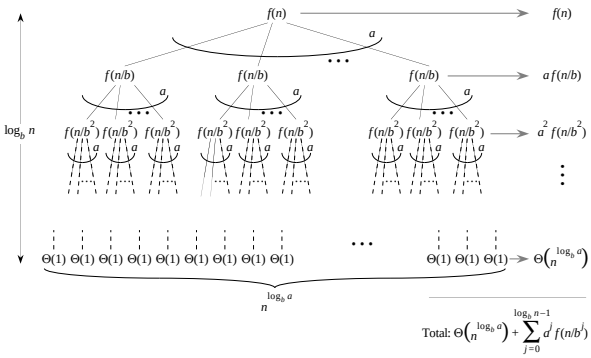
III. Master method

For $T(n) = aT(n/b) + f(n)$,
$a \geq 0$ and $b > 0$, and f asymptotically positive

Abstractly: $T(n)$ is runtime for an algorithm;
it's unknown, but we do know that:

- a subproblems of size n/b are solved recursively, each in time $T(n/b)$.
- $f(n)$ is the cost of dividing problem (beforehand) and combining the results (afterward).

Recursion tree for $T(n) = aT(n/b) + f(n)$:



Split problem into a parts at $\log_b n$ levels:

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

Analysis

\# of leaves $= a^{\log_b n} = n^{\log_b a}$
Iterating the recurrence yields

$$\begin{aligned} T(n) &= f(n) + aT(n/b) \\ &= f(n) + af(n/b) + a^2 T(n/b^2) \\ &= f(n) + af(n/b) + a^2 f(n/b^2) + \cdots \\ &\quad + a^{\log_b n - 1} f(n/b^{\log_b n - 1}) + a^{\log_b n} T(1) \ . \end{aligned}$$

Thus,

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

First term is total division/recombination cost;
Second term is cost of doing all size-1 subproblems.

Intuition:

Three common cases:
Running time dominated by cost at leaves
Running time evenly distributed throughout tree
Running time dominated by cost at root
So: recurrence solution determined by dominant term!

Three common cases: Compare $f(n)$ and $n^{\log_b a}$

Case 1:
$f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$.

- $f(n)$ grows polynomially **slower** than $n^{\log_b a}$ (by factor n^{ϵ})
- Weight of each level increases geometrically from root to leaves $(1, a^1, a^2, \cdots)$ (thus decreases geometrically from leaves to root)
- Leaves contain constant fraction of total weight; i.e., total is constant $\times$ \# of leaves. (What are these?)

Since we need only determine T to within a constant, computing work done at leaves is sufficient!

Analysis of Case 1:

$$\begin{aligned}
\sum_{j=0}^{\log_b n-1} a^j f(n/b^j) &= O\left(\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a-\epsilon}\right) \\
&= O\left(n^{\log_b a-\epsilon} \sum_j \left(\frac{ab^\epsilon}{b^{\log_b a}}\right)^j\right) \\
&= O\left(n^{\log_b a-\epsilon} \sum_j b^{\epsilon j}\right) \\
&= O\left(n^{\log_b a-\epsilon}\left(\frac{b^{\epsilon \log_b n}-1}{b-1}\right)\right) \\
&= O\left(n^{\log_b a}\right)
\end{aligned}$$

Where again we used

$$\sum_{k=0}^{N} x^k = \frac{x^{n+1}-1}{x-1}$$

Thus $\boxed{T(n) = O\left(n^{\log_b a}\right)}$

Case 2:

$f(n) = \Theta(n^{\log_b a} \lg^k n)$, for some constant $k \geq 0$

- $f(n)$ and $n^{\log_b a}$ same to within a polylogarithmic factor
- Weight decreases (arithmetically if $k = 0$)

$\boxed{T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)}$

Case 3:

$f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$
also, need "regularity condition":
$\exists c < 1$ and $n_0 > 0$ s.t. $af(n/b) \leq cf(n)\ \forall n > n_0$

- $f(n)$ grows polynomially faster than $n^{\log_b a}$
- Weight is geometrically decreasing
- Root contains constant fraction of total weight (i.e., total is constant $\times$ weight of root)

$\boxed{T(n) = \Theta(f(n))}$

Master Theorem:

Given recurrence of form $T(n) = aT(n/b) + f(n)$

1. $f(n) = O(n^{\log_b a-\epsilon})$
   $\Rightarrow T(n) = O\left(n^{\log_b a}\right)$
2. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some $k \geq 0$
   $\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$
3. $f(n) = \Omega(n^{\log_b a+\epsilon})$ and
   $af(n/b) \leq cf(n) \Rightarrow$ for some $c < 1, n > n_0$
   $\Rightarrow T(n) = \Theta(f(n))$

Careful: master method can't solve *every* recurrence of this form! (Examples later.)

MT examples

Merge sort:
$T(n) = 2T(n/2) + \Theta(n); a = 2, b = 2$

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

MT examples

Merge sort:
$T(n) = 2T(n/2) + \Theta(n); a = 2, b = 2$

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

$$\Theta(n)/n = \Theta(1) = \Theta(\lg^0 n)$$

$$\Rightarrow \text{Case 2, with } k = 0$$

$$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n) = \Theta(n \lg n)$$

MT examples

$T(n) = 7T(n/2) + \Theta(n^2)$
(Recurrence for Strassen Algorithm, next lecture)

$$n^{\log_b a} = n^{\log_2 7} = n^{\lg 7}$$

$$\Theta(n^2)/n^{\lg 7} = \Theta(n^{2-\lg 7}) = O(n^{-.8})$$

$$\Rightarrow \text{Case 1}$$

$$T(n) = \Theta(n^{\lg 7})$$

MT examples

Binary **search** (not sort!):
$T(n) = T(n/2) + \Theta(1); (a = 1, b = 2)$

$$n^{\lg 1} = n^0 = 1$$

$$\Theta(1)/1 = \Theta(\lg^0 n) \Rightarrow \text{Case 2}$$

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

MT examples

$$T(n) = 4T(n/2) + n^3; (a = 4, b = 2)$$
$$n^3/n^{\lg 4} = n \Rightarrow \text{Case 3}$$
$$T(n) = \Theta(n^3)$$

Are we done?

MT examples

$$T(n) = 4T(n/2) + n^3; (a = 4, b = 2)$$
$$n^3/n^{\lg 4} = n \Rightarrow \text{Case 3}$$
$$T(n) = \Theta(n^3)$$

Regularity condition:

$$4f(n/2) \le cf(n)$$
$$4n^3/8 \le cn^3$$
$$n^3/2 \le cn^3$$
$$c = 3/4 < 1$$

More MT examples on recitation tomorrow

One more recurrence:

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

$$\frac{n^2/\lg n}{n^2} = 1/\lg n = \begin{cases} O(n^{-\epsilon}), & \epsilon > 0? \lg n = \Omega(n^\epsilon)? \text{ No} \\ \Theta(\lg^k n), & k \ge 0? \text{ No} \\ \Omega(n^\epsilon), & \epsilon > 0? \text{ No} \end{cases}$$

No case applies!

What's going on?

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

$$\frac{n^2/\lg n}{n^2} = 1/\lg n = \begin{cases} O(n^{-\epsilon}), & \epsilon > 0? \lg n = \Omega(n^\epsilon)? \text{ No} \\ \Theta(\lg^k n), & k \ge 0? \text{ No} \\ \Omega(n^\epsilon), & \epsilon > 0? \text{ No} \end{cases}$$

Gap between Cases 1 & 2:
$f(n)$ smaller than $n^{\log_b a}$, but not *polynomially* smaller!
Gap between Cases 2 & 3:
$f(n)$ larger than $n^{\log_b a}$, but not *polynomially* larger!
(Or, regularity condition can fail)

(Solution: $T(n) = \Theta(n^2 \lg \lg n)$ — substitution)