Portland State University

W'21 CS 584/684 Algorithm Design & Analysis

Fang Song

Lecture 3

- Exponentiation
- Solving recurrences
- Graph basics

Review: Divide-&-Conquer

1. **Divide**
   - Divide the given instance of the problem into several independent smaller instances of the same problem.
2. **Delegate**
   - Solve smaller instances recursively, i.e., delegate each smaller instance to the Recursion Fairy.
3. **Combine**
   - Combine solutions of smaller instance into the final solution for the given instance.

Exponentiation

Given: integers a, b. b is n-bit long.

Goal: $c = a^b$.

How many multiplications?

- Naive algorithm: $\Theta(b) = \Theta(2^n)$.
  - Exponential in the input length!
- Divide-&-Conquer
  - Linear in the input length!

1 subproblem only

(Not 2 or more)

$$a^b = \begin{cases} \boxed{a^{b/2} \cdot a^{b/2}}, & \text{if } b \text{ even} \\ \boxed{a^{(b-1)/2} \cdot a^{(b-1)/2}} \cdot a, & \text{if } b \text{ odd} \end{cases}$$

$$T(b) = T(b/2) + O(1) = O(\log b) = O(n)$$

Recurrences

- **Definition: an equation or inequality that describes a function in terms of its values on smaller inputs.**
  - Sloppiness: ignore floor/ceilings; $T(1) = O(1)$

    Example. $T(n) = \begin{cases} O(1), & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n), & \text{if } n > 1 \end{cases}$

- **Recurrences we have seen.**
  - Merge sort: $T(n) = 2T(n/2) + O(n) = O(n^2)$
  - Karatsuba's integer multiplication: $T(n) = 3T(n/2) + O(n) \approx O(n^{1.59})$
  - Strassen's matrix multiplication: $T(n) = 7T(n/2) + O(n^2) \approx O(n^{2.81})$
  - Exponentiation: $T(b) = T(b/2) + O(1) = O(\log b) = O(n)$

Method #1: Recursion Tree

1. **Form recursion tree to guess a solution.**
    - Draw the tree of recursive calls.
    - Each node gets assigned the work done during that call to the procedure (dividing and combining).
    - Total work is sum of work at all nodes.

2. **Prove it by induction.**

Recursion tree for Mergesort

$T(n) = 2T(n/2) + n$ Ignore floor/ceil & constant factor in merging time $O(n)$.

1 **Draw tree of recursive calls**

Recursion tree for Mergesort

$T(n) = 2T(n/2) + n$ Ignore floor/ceil & constant factor in merging time $O(n)$.

2 **Assign work at each level (dividing and combining)**

Recursion tree for Mergesort

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

Ignore floor/ceil & constant factor in merging time $O(n)$.

3 **Total work = sum of all nodes**

Method #2: Master theorem

- A “cookbook” for solving recurrences of the form

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

  - $a \geq 1, b > 1$.
  - f asymptotically positive: $\exists n_0 > 0$, s.t. $f(n) > 0, \forall n > n_0$.

- 3 typical cases depending on $f(n)$ vs. $n^{\log_b a}$

	$T(n)$	$f(n)$ vs. $n^{\log_b a}$
1	$\Theta(n^{\log_b a})$	$f(n) = O(n^{(\log_b a)-\epsilon})$ for some $\epsilon > 0$.
2	$\Theta(n^{\log_b a} \log n)$	$f(n) = O(n^{\log_b a})$
3	$\Theta(f(n))$	$f(n) = \Omega(n^{(\log_b a)+\epsilon})$ for some $\epsilon > 0$, and $f(n/b) \leq cf(n)$ for some $c < 1$.

1. f grows slower by a polynomial n^ϵ factor

2. Grow at “same” rate.

3. f grows poly-faster + regularity condition.

Master theorem in use

	$T(n)$	$f(n)$ vs. $n^{\log_b a}$
1	$\Theta(n^{\log_b a})$	$f(n) = O(n^{(\log_b a)-\epsilon})$ for some $\epsilon > 0$.
2	$\Theta(n^{\log_b a} \log n)$	$f(n) = O(n^{\log_b a})$
3	$\Theta(f(n))$	$f(n) = \Omega(n^{(\log_b a)+\epsilon})$ for some $\epsilon > 0$, and $f(n/b) \leq cf(n)$ for some $c < 1$.

- **In-class exercise: solve these by master theorem.**
  1. Merge sort: $T(n) = 2T(n/2) + O(n)$.
  2. Karatsuba's integer multiplication: $T(n) = 3T(n/2) + O(n)$.
  3. Strassen's matrix multiplication: $T(n) = 7T(n/2) + O(n^2)$.
  4. Exponentiation: $T(n) = T(n/2) + O(1)$.
  5. $T(n) = 4T(n/2) + O(n^3)$.

Master theorem in use

	$T(n)$	$f(n)$ vs. $n^{\log_b a}$
1	$\Theta(n^{\log_b a})$	$f(n) = O(n^{(\log_b a)-\epsilon})$ for some $\epsilon > 0$.
2	$\Theta(n^{\log_b a} \log n)$	$f(n) = O(n^{\log_b a})$
3	$\Theta(f(n))$	$f(n) = \Omega(n^{(\log_b a)+\epsilon})$ for some $\epsilon > 0$, and $f(n/b) \le cf(n)$ for some $c < 1$.

1. $T(n) = 2T(n/2) + O(n)$. [Merge sort]
   - $a = 2, b = 2, n^{\log_b a} = n = f(n)$. Case 2: $T(n) = O(n \log n)$
2. $T(n) = 3T(n/2) + O(n)$. [Karatsuba's integer multiplication]
   - $a = 3, b = 2, n^{\log_b a} = n^{\log_2 3} \approx n^{1.58}$. $f(n) = n = O(n^{1.58-\epsilon})$ for $\epsilon = 0.5$.
   - Case 1: $T(n) = O(n^{\log_b a})$.

Master theorem in use, cont'd

	$T(n)$	$f(n)$ vs. $n^{\log_b a}$
1	$\Theta(n^{\log_b a})$	$f(n) = O(n^{(\log_b a)-\epsilon})$ for some $\epsilon > 0$.
2	$\Theta(n^{\log_b a} \log n)$	$f(n) = O(n^{\log_b a})$
3	$\Theta(f(n))$	$f(n) = \Omega(n^{(\log_b a)+\epsilon})$ for some $\epsilon > 0$, and $f(n/b) \leq cf(n)$ for some $c < 1$.

3. $T(n) = 7T(n/2) + O(n^2)$. [Strassen's matrix multiplication]
   - $a = 7, b = 2, n^{\log_b a} = n^{\log_2 7} \approx n^{2.81}$. $f(n) = n^2 = O(n^{2.81-\epsilon})$ for $\epsilon = 0.8$.
   - Case 1: $T(n) = O(n^{\log_2 7}) \approx O(n^{2.81})$.
4. $T(n) = T(n/2) + O(1)$. [Exponentiation]
   - $a = 1, b = 2, n^{\log_b a} = n^0 = 1. f(n) = O(1)$. Case 2: $T(n) = O(\log n)$.

Master theorem in use, cont'd

	$T(n)$	$f(n)$ vs. $n^{\log_b a}$
1	$\Theta(n^{\log_b a})$	$f(n) = O(n^{(\log_b a)-\epsilon})$ for some $\epsilon > 0$.
2	$\Theta(n^{\log_b a} \log n)$	$f(n) = O(n^{\log_b a})$
3	$\Theta(f(n))$	$f(n) = \Omega(n^{(\log_b a)+\epsilon})$ for some $\epsilon > 0$, and $f(n/b) \leq cf(n)$ for some $c < 1$.

5. $T(n) = 4T(n/2) + O(n^3)$.
   - $a = 4, b = 2, n^{\log_b a} = n^2, f(n) = n^3 = \Omega(n^{2+\epsilon})$ for $\epsilon = 1$.
   - Check regularity condition: $f(n/b) = (n/2)^3 \leq cn^3$ for $c = 0.5 < 1$.
   - Case 3: $T(n) = \Theta(n^3)$.

Master theorem doesn't solve it all

	$T(n)$	$f(n)$ vs. $n^{\log_b a}$
1	$\Theta(n^{\log_b a})$	$f(n) = O(n^{(\log_b a)-\epsilon})$ for some $\epsilon > 0$.
2	$\Theta(n^{\log_b a} \log n)$	$f(n) = O(n^{\log_b a})$
3	$\Theta(f(n))$	$f(n) = \Omega(n^{(\log_b a)+\epsilon})$ for some $\epsilon > 0$, and $f(n/b) \leq cf(n)$ for some $c < 1$.

- ✳ Solve $T(n) = 4T(n/2) + n^2/\log n$.
  - $a = 4, b = 2, n^{\log_b a} = n^2, f(n) = n^2/\log n$.
  - Master theorem doesn't apply! For any constant $\epsilon > 0, n^\epsilon = \omega(\log n)$.
- ◉ Generalization exists.
  - E.g. Akra-Bazzi method https://en.wikipedia.org/wiki/Akra%E2%80%93Bazzi_method

Master theorem: proof idea

Graph basics

Scratch