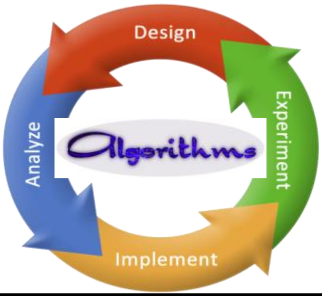
Algorithm Design and Analysis

Victor Adamchik — CS 15-451 — Spring 2015

Lecture 1 — Jan 12, 2015 — Carnegie Mellon University

Intro to Algorithms



Outline

1. Administria
2. The Master Theorem
3. Karatsuba's Algorithm

Course Staff

Victor Adamchik

Danny Sleator

TAs: TBA

Web Sites

www.cs.cmu.edu/afs/cs/academic/class/15451-s15

Calendar, Slides, Notes, Homeworks, Course Policy, Grades, …

http://piazza.com/

Questions, Comments, Announcements, …

Textbook

There is no textbook.

Slides will be posted on the website.

Some supplementary notes will also be posted.

Grading

30%	Homework	(weekly, written and oral)
10%	Quizzes	(weekly)
30%	Tests	(2 midterms)
30%	Final	

Homework

Homeworks roughly every week

Approx: 8 written and 3 oral

4 late days for written Hwks
2 late days at most per Hwk

We will drop the lowest written Hwk

Collaboration

You may work in a group of ≤ 3 people.

You *must* report who you worked with.

You must think about *each of the problems* by yourself for ≥ 30 minutes before discussing them with others.

You must write up *all* solutions by yourself.

Cheating

You MAY NOT

Share written work.

Get help from anyone besides your collaborators, staff.

Refer to solutions/materials from earlier versions of 451 or the web

Quizzes

Every week, online

Tested on material from the previous 2-3 lectures.

These are designed to be easy, assuming you are keeping up with the lectures.

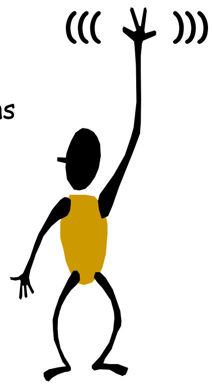
Midterm Tests

There will be TWO tests given in class.

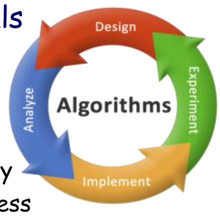
Designed to be doable...

"Semi-cumulative."

Feel free to ask questions



Course Goals



1. Understand
   a) Algorithms
   b) Design techniques
2. Analyze algorithm efficiency
3. Analyze algorithm correctness
4. Communicate about code
5. Design your own algorithm

Divide and Conquer
(review of 15-210)

A divide-and-conquer algorithm consists of
- dividing a problem into smaller subproblems
- solving (recursively) each subproblem
- then combining solutions to subproblems to get solution to original problem

Runtime

Suppose $T(n)$ is the number of steps in the worst case needed to solve the problem of size n.
Let us split a problem into $a>1$ subproblems, each of which is of the input size n/b where $b>1$.

$T(n) = 2T(n/2) + n$ — Merge sort

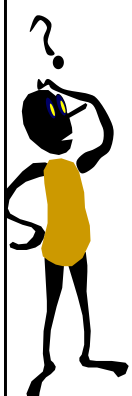
$T(n) = T(n/2) + 1$ — Binary search

The recurrences have some initial conditions

Runtime

The total complexity $T(n)$ is obtained by all steps needed to solve smaller subproblems $T(n/b)$ plus the work needed $f(n)$ to combine solutions into a final one.

$$T(n) = a \cdot T(n/b) + f(n)$$



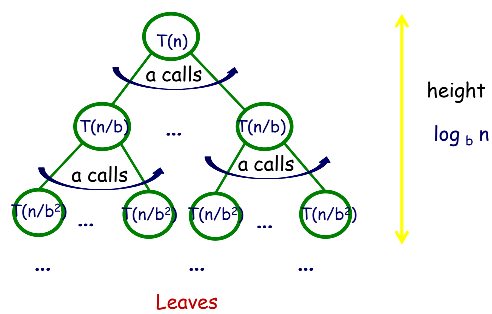
$$T(n) = a \cdot T(n/b) + f(n)$$

How do we solve this recurrence?

Tree of Recursive Calls !

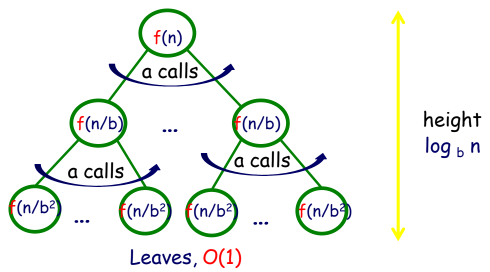
Tree method: $T(n) = a \cdot T(n/b) + f(n)$

Draw a tree of recursive calls:

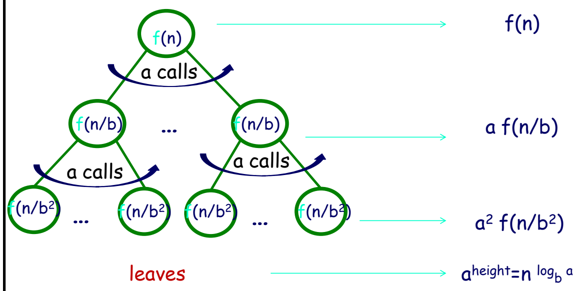


Tree method: $T(n) = a \cdot T(n/b) + f(n)$

This tree represents the total work:



$T(n) = a \cdot T(n/b) + f(n) \qquad T(1) = O(1)$



Constant work at leaves!!

The Master Theorem

$$T(n) = T(1)\, n^{\log_b a} + \sum_{k=0}^{h-1} a^k f\left(\frac{n}{b^k}\right)$$

where $h = \log_b n$

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{Leaves dominate} \\ \Theta(n^{\log_b a} \log^p n) & \text{Both} \\ \Theta(f(n)) & \text{Internal nodes dominate} \end{cases}$$

It (all) depend on the function f(x) – a combining step

The Master Theorem

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), \text{if } f(n) \in O(n^{\log_b a - \delta}) \\ \Theta(n^{\log_b a} \log^p n), \text{if } f(n) \in \Theta(n^{\log_b a} \log^{p-1} n) \\ \Theta(f(n)), \text{if } f(n) \in \Omega(n^{\log_b a + \delta}) \end{cases}$$

for some constant $\delta > 0$ and $\delta \to 0$

and constant p = 1, 2, …

Case I

if $f(n) \in O(n^{\log_b a - \delta})$, then $T(n) = \Theta(n^{\log_b a})$

Proof. The solution to the recurrence is

$$T(n) = \Theta(n^{\log_b a}) + \sum_{k=0}^{h-1} a^k f\left(\frac{n}{b^k}\right)$$

We simplify the sum in the rhs

$$\sum_{k=0}^{h-1} a^k f\left(\frac{n}{b^k}\right) \le c \sum_{k=0}^{h-1} a^k \left(\frac{n}{b^k}\right)^{\log_b a - \delta} = c\, n^{\log_b a - \delta} \sum_{k=0}^{h-1} \left(\frac{a}{b^{\log_b a}}\right)^k b^{\delta k} =$$

$$= c\, n^{\log_b a - \delta} \sum_{k=0}^{h-1} b^{\delta k} \le c\, n^{\log_b a - \delta} \sum_{k=0}^{\infty} b^{\delta k} \le c_1\, n^{\log_b a - \delta}$$

since $b^{\delta} < 1$. It follows that $T(n) = \Theta(n^{\log_b a})$ QED

Case II

if $f(n) \in \Theta(n^{\log_b a} \log^{p-1} n)$, then $\Theta(n^{\log_b a} \log^p n)$

Proof. We prove this for p=1. The solution to the recurrence is

$$T(n) = \Theta(n^{\log_b a}) + \sum_{k=0}^{h-1} a^k f\left(\frac{n}{b^k}\right)$$

We simplify the sum in the rhs

$$\sum_{k=0}^{h-1} a^k f\left(\frac{n}{b^k}\right) = \sum_{k=0}^{h-1} a^k \left(\frac{n}{b^k}\right)^{\log_b a} = n^{\log_b a} \sum_{k=0}^{h-1} 1 =$$

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

It follows that

$$T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \log_b n) = \Theta(n^{\log_b a} \log n) \quad \text{QED}$$

Example - 1

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) \\ \Theta(n^{\log_b a} \log^p n) \\ \Theta(f(n)) \end{cases}$$

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

Work at leaves is $n^{\log_b a} = n^{\log_2 4} = n^2$

$f(n) = n$ $\quad f(n) = O(n^2)$

It follows, $T(n) \in \Theta(n^2)$

Example - 2

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) \\ \Theta(n^{\log_b a} \log^p n) \\ \Theta(f(n)) \end{cases}$$

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

Work at leaves is $n^{\log_b a} = n^{\log_2 4} = n^2$

$f(n) = n^2$ $\quad f(n) \in \Theta(n^2)$

It follows, $T(n) \in \Theta(n^2 \log n)$

Example - 3

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) \\ \Theta(n^{\log_b a} \log^p n) \\ \Theta(f(n)) \end{cases}$$

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

Work at leaves is $n^{\log_b a} = n^{\log_2 4} = n^2$

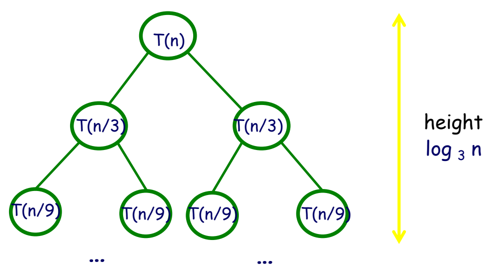
$f(n) = n^3$ $\quad f(n) \in \Omega(n^2)$

It follows, $T(n) \in \Theta(n^3)$

Example:

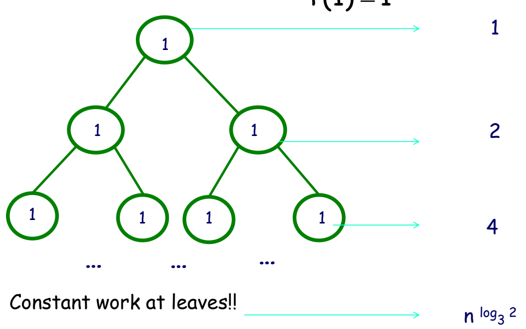
$$T(n) = 2T(n/3) + 1$$
$$T(1) = 1$$

Draw a tree of recursive calls:



Example:

$$T(n) = 2T(n/3) + 1$$
$$T(1) = 1$$



Example:

$$T(n) = 2T(n/3) + 1$$
$$T(1) = 1$$

$$T(n) = n^{\log_3 2} + \sum_{k=0}^{h-1} 2^k$$

$$T(n) = n^{\log_3 2} + 2^h - 1$$

height $h = \log_3 n$

$$T(n) = -1 + 2 * n^{\log_3 2}$$

Karatsuba's Algorithm (1962)

Fast integer multiplication

Integer Multiplication

Given two n-digit integers.
Using a grammar school approach,
we can multiply them in $\Theta(n^2)$ time.

Observe, any integer can be split into two parts

$154517766 = 15451 * 10^4 + 7766$

Integer Multiplication: divide-and-conquer

$num_1 = x_1 * 10^p + x_0$

$num_2 = y_1 * 10^p + y_0$

p=n/2

x_1	x_0
y_1	y_0

$num_1 * num_2 = x_1 * y_1 * 10^{2p} + (x_1 * y_0 + x_0 * y_1) * 10^p + x_0 * y_0$

The worst-case complexity:

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

by the master theorem $T(n) = \Theta(n^2)$

Karatsuba's Algorithm

$num_1 * num_2 = x_1 * y_1 * 10^{2p} + \underline{(x_1 * y_0 + x_0 * y_1)} * 10^p + x_0 * y_0$

$num_1 * num_2 = x_1 * y_1 * 10^{2p} +$
$\underline{((x_1 + x_0) * (y_1 + y_0) - x_1 * y_1 - x_0 * y_0)} * 10^p + x_0 * y_0$

The worst-case complexity:

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

by the master theorem $T(n) = \Theta(n^{\log 3}) = \Theta(n^{1.58})$

3-way splitting

The key idea is to divide a large integer into 3 parts (rather than 2) of size approximately n/3 and then multiply those parts.

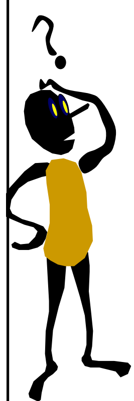
This is similar to 3-way merging.

The worst-case:
(x is unknown)

$T(n) = x \cdot T(n/3) + O(n)$

by the master theorem $T(n) = \Theta(n^{\log_3 x}) = O(n^{1.58})$

$\log_3 x < 1.58$ $x = 5$ Thus we need to reduce 9 mults to 5



$T(n) = 5T(n/3) + O(n)$

Is it possible to reduce a number of multiplications from 9 to 5?

3-way split
T. Cook (1966)

x_2	x_1	x_0

y_2	y_1	y_0

$Z_0 = x_0\, y_0$
$Z_1 = (x_0+x_1+x_2)\,(y_0+y_1+y_2)$
$Z_2 = (x_0+2\,x_1+4\,x_2)\,(y_0+2\,y_1+4\,y_2)$
$Z_3 = (x_0-x_1+x_2)\,(y_0-y_1+y_2)$
$Z_4 = (x_0-2\,x_1+4\,x_2)\,(y_0-2\,y_1+4\,y_2)$

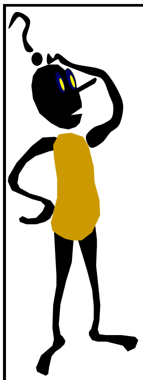
Further Generalization:
k-way split

splits	Number of multiplications
2	3
3	5
4	7

$$T(n) = (2k-1)T(n/k) + n$$

$$T(n) = n^{\log_k(2k-1)}$$

$n^{1.58}$, $n^{1.46}$, $n^{1.40}$, $n^{1.36}$, $n^{1.33}$, $n^{1.31}$, $n^{1.30}$, $n^{1.28}$...

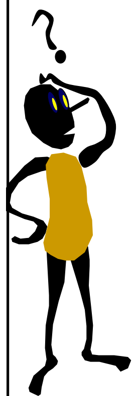


$$T(n) = n^{\log_k(2k-1)}$$

$n^{1.58}$, $n^{1.46}$, $n^{1.40}$, $n^{1.36}$, $n^{1.33}$, $n^{1.31}$, $n^{1.30}$, $n^{1.28}$...

Is it possible to multiply two integers in <u>linear</u> time?

$$\log_k(2k-1) = \frac{\ln(2k-1)}{\ln k} = 1 + \frac{\ln(2-1/k)}{\ln k} > 1+\varepsilon$$



$$T(n) = n^{\log_k(2k-1)}$$

Is it always possible to reduce k^2 multiplications to $2k-1$?

Is it always possible to reduce k^2 multiplications to 2k-1?

Consider k-way split

$polyn_1 = a_{k-1}\, x^{k-1} + a_{k-2} * x^{k-2} + \ldots + a_1 * x + a_0$
$polyn_2 = b_{k-1}\, x^{k-1} + b_{k-2} * x^{k-2} + \ldots + b_1 * x + b_0$

$polyn_1 * polyn_2 = a_{k-1}\, b_{k-1} * x^{2k-2} + \ldots + (a_1\, b_0 + b_1\, a_0) * x + a_0\, b_0$

It has 2k-1 coefficients, which uniquely define a polynomial. Therefore, it requires 2k-1 new variables, thus we should have at least 2k-1 multiplications. But that is not simple to find them...

Multiplication of large integers of n digits can be done in time
O(n log n log log n)
thanks to the Fast Fourier Transform.