Analysis of Algorithms

V. Adamchik

Lecture 6

CSCI 570

University of Southern California

Divide-And-Conquer Algorithms

Reading: chapter 5

Divide and Conquer Algorithms

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

Binary Search

Given a sorted array of size n:

- compare the search item with the middle
- if it's less, search in the lower half
- if it's greater, search in the upper half
- if it's equal or the entire array has been searched, terminate.

$$\underbrace{1\ \ 2\ \ 3}_{n/2}\ \ 4\ \ \overbrace{5\ \ 6\ \ 7\ \ 8}^{n/2}$$

linear	binary
n	n
$n-1$	$n/2$
$n-2$	$n/4$
$\vdots$	$\vdots$
	$\Omega(n)$

Mergesort

divides an unsorted list into two equal or nearly equal sub lists

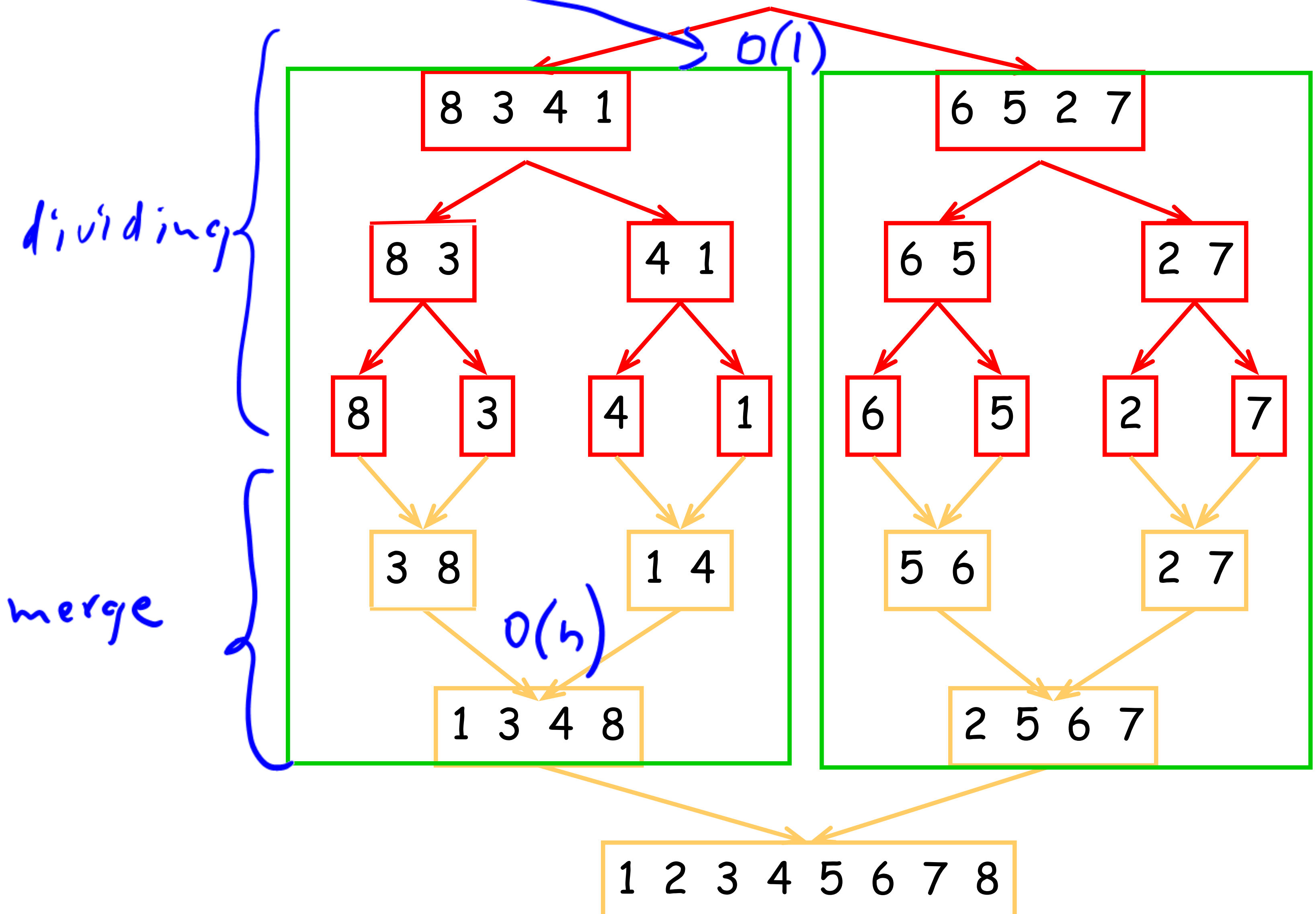
sorts each of the sub lists by calling itself recursively, and then

merges the two sub lists together to form a sorted list

$$\underbrace{8\ \ 3\ \ 4\ \ 1}_{n/2}\ \Big|\ \overbrace{6\ \ 5\ \ 2\ \ 7}^{n/2}$$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(1) + O(n)$$

$$T(1) = 1$$



D&C Recurrences

Suppose $T(n)$ is the number of steps in the worst case needed to solve the problem of size n.

We define the runtime complexity $T(n)$ by a recurrence equation.

Binary Search: $T(n) = 1 \cdot T\left(\frac{n}{2}\right) + \underbrace{O(1)}_{\text{split.}} + \underbrace{O(1)}_{\text{comp.}}$

MergeSort: $T(n) = \underbrace{2 \cdot T\left(\frac{n}{2}\right)}_{\text{dividing}} + \underbrace{O(1)}_{\text{split}} + \underbrace{O(n)}_{\text{merge}}$

D&C Recurrences

Suppose $T(n)$ is the number of steps in the worst case needed to solve the problem of size n.

Let us divide a problem into $a \geq 1$ subproblems, each of which is of the input size n/b where $b>1$.

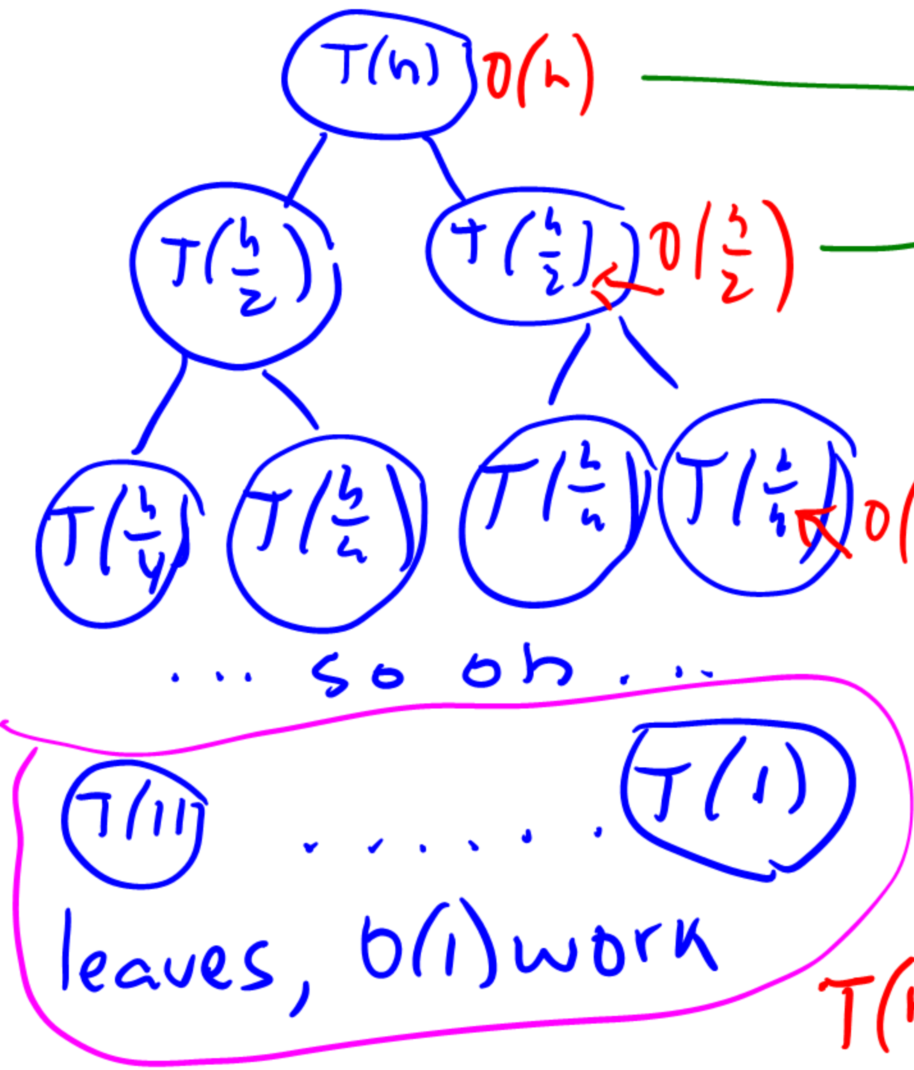
The total complexity $T(n)$ is obtained by

function ≥ 0

$$\boxed{T(n) = \underbrace{a \cdot T(n/b)}_{\text{dividing}} + \underbrace{f(n)}_{\text{conquering}}}$$

Here $f(n)$ is a complexity of combining subproblem solutions (including complexity of *dividing* step).

Mergesort: tree of recursive calls



n

$2 \cdot \frac{n}{2} = n$

$4 \cdot \frac{n}{4} = n$

. . . .

$\sum \text{leaves} = n$

$T(n) = \overbrace{n + n + \dots + n}^{\text{height}} = n \log n$

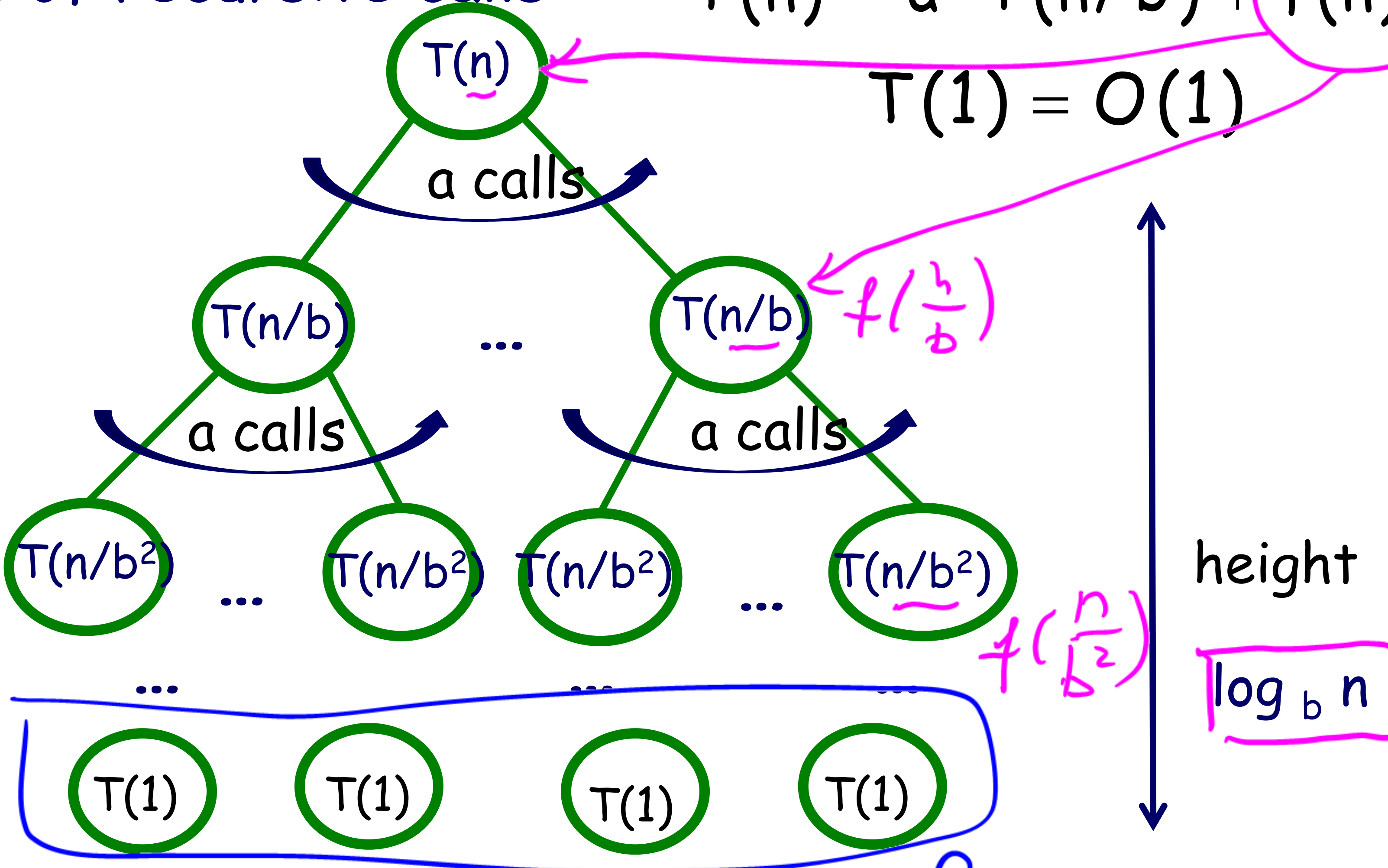
$h = \log n$

$\boxed{2^h}$

Tree of recursive calls

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

$$T(1) = O(1)$$



leaves. How many leaves?

Counting leaves

h = height

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

$$a^h = a^{\log_b n} = n^{\frac{1}{\log_a b}} =$$

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

$$\log_b n = \frac{\log_a n}{\log_a b}$$

Discussion Problem 1

1 Draw a recursive tree for:

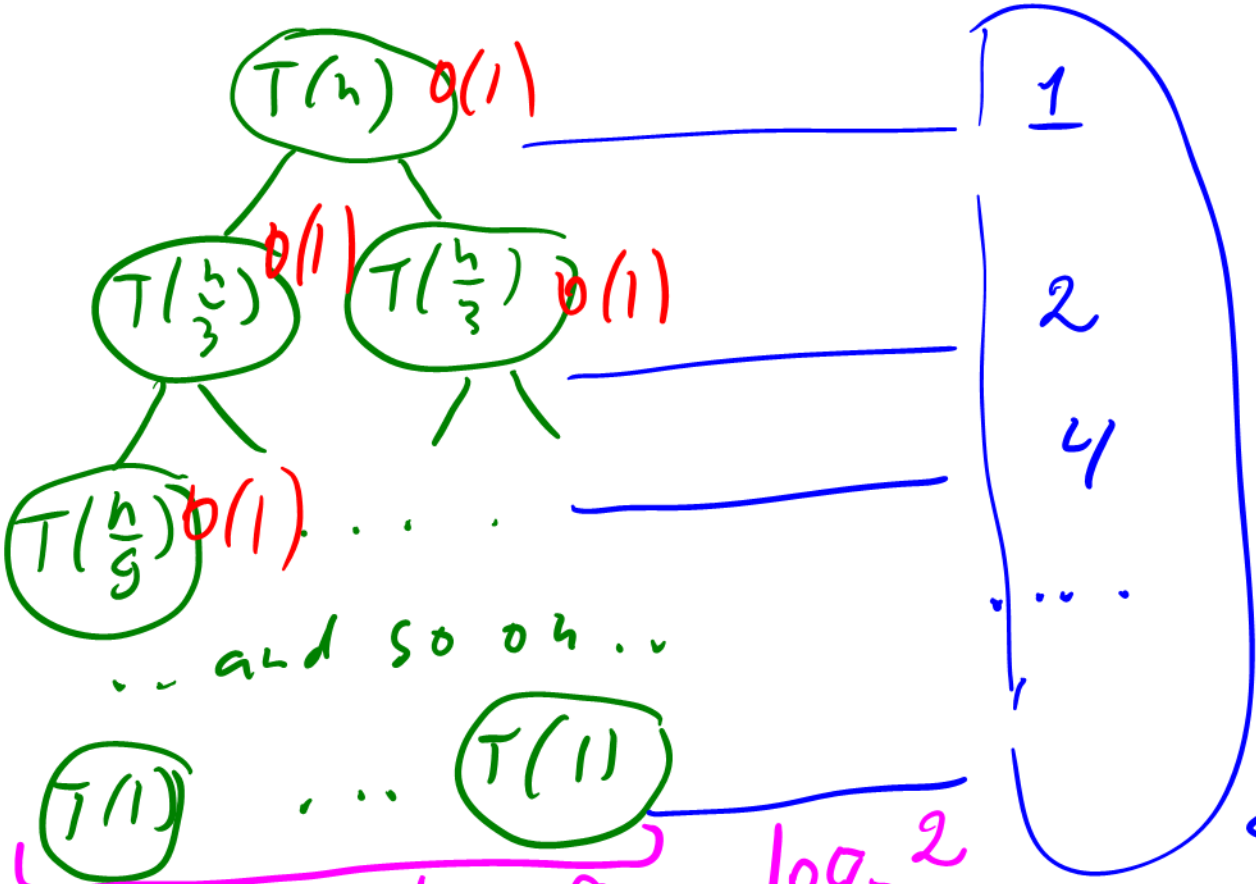
$$T(n) = 2 \cdot T(n/3) + O(1)$$

$$T(1) = 1$$

$a = 2$; $b = 3$; $f(n) = O(1)$

2 and compute the total work $T(n)$.

Solution



sum it up

leaves: $n^{\log_b a} = n^{\log_3 2}$

$$T(n) = \underbrace{1+2+4+\dots+2^{h-1}}_{\text{internal nodes}} + \underbrace{n^{\log_3 2}}_{\text{leaves}} = O\left(n^{\log_3 2}\right)$$

it depends on f(n)

free of f(n)

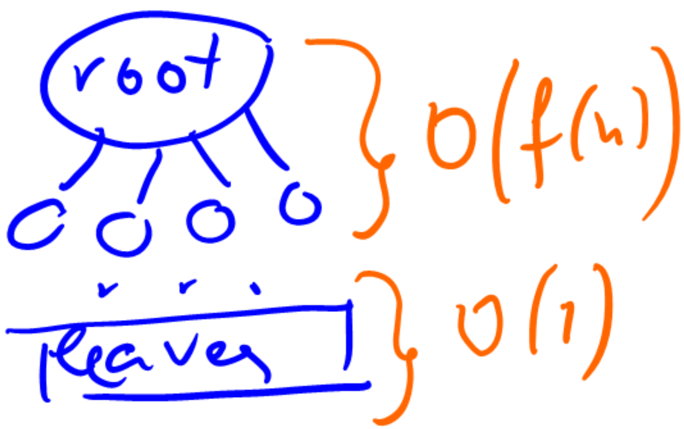
verify!

The Master Theorem

The master method provides a straightforward ("cookbook") method for solving recurrences of the form

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

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is a <u>positive</u> function.



$$T(n) = \sum O(f(n)) + \sum O(1)$$

The Master Theorem

≥ 0 constants

$$T(n) = a \cdot T(n/b) + f(n), \quad \boxed{a \geq 1 \text{ and } b > 1}$$

Let $c = \log_b a$.

Case 1: (only leaves)

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

Case 2: (all nodes) Mergesort ($k = 0$)

if $f(n) = \Theta(n^c \log^k n)$, $k \geq 0$, then $T(n) = \Theta(n^c \log^{k+1} n)$

!!   ?!!

Case 3: (only internal nodes)

if $f(n) = \Omega(n^{c+\varepsilon})$, then $T(n) = \Theta(f(n))$ for some $\varepsilon > 0$.

Discussion Problem 2

Solve the recurrence by the Master Theorem:

$$T(n) = 16\,T(n/4) + \underbrace{5\,n^3}_{f(n)}$$

$a = 16$
$b = 4$
$c = \log_b a = \log_4 16 = 2$

leaves $\quad f(n)$

$O(n^2) \; ? \; O(n^3)$

$$\boxed{T(n) = \Theta(n^3)}$$

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

Case 1: if $f(n)=O(n^{c-\varepsilon})$, then $T(n)=\Theta(n^c)$
Case 2: if $f(n)=\Theta(n^c \log^k n)$, then $T(n)=\Theta(n^c \log^{k+1} n)$
$\boxed{\text{Case 3:}}$ if $f(n)=\Omega(n^{c+\varepsilon})$, then $T(n) = \Theta(f(n))$

where $\boxed{c = \log_b a.}$

Discussion Problem 3

Solve the recurrence by the Master Theorem:

case 1 leaves $f(n)$

1. $A(n) = 3\ A(n/3) + 15$, $O(n)$ $O(1)$, $A(n) = \Theta(n)$

case 3 leaves $f(n)$

2. $B(n) = 4\ B(n/2) + n^3$, $O(n^2)$ $O(n^3)$, $B(n) = \Theta(n^3)$

case 2, $k = 0$

3. $C(n) = 4\ C(n/2) + n^2$, $C(n) = \Theta(n^2 \log n)$

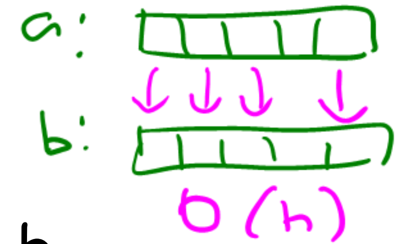
4. $D(n) = 4\ D(n/2) + n$, $D(n) = \Theta(n^2)$

case 2 with $k = 1$

5. $E(n) = 4 \cdot E\left(\frac{n}{2}\right) + n^2 \log n$, $E(n) = \Theta(n^2 \log^2 n)$

Integer Multiplication

$a+b$



Given two n-digit integers a and b, compute a × b.

$\boxed{\text{Brute force}}$ solution: $\boxed{O(n^2)}$ bit operations.

$$\underbrace{15451|7766}_{n \text{ digits}} = 15451 \cdot 10^4 + 7766$$

$\frac{n}{2}$ digits $\qquad \frac{n}{2}$

$$\underbrace{a \cdot b}_{n \text{ bits}} = (x_1 \cdot 10^{n/2} + x_0) \cdot (y_1 \cdot 10^{n/2} + y_0) =$$

$$= \underbrace{x_1 y_1}_{\frac{n}{2} \text{ bits}} \cdot \underbrace{10^n}_{\substack{\text{base} \\ O(1)}} \oplus \underbrace{(x_1 y_0 \oplus x_0 y_1)}_{\text{rewrite}} \underbrace{10^{n/2}}_{\substack{\text{base} \\ O(1)}} \oplus x_0 y_0$$

additions

$O\left(\frac{n}{2}\right) = O(n)$

$$T(n) = \boxed{4} \cdot T\left(\frac{n}{2}\right) + O(n)$$

$$\boxed{T(n) = \Theta(n^2)}$$

case 1

$$\begin{array}{r} 1234 \\ \times \quad \underline{1111} \\ 1234 \\ 1234 \\ 1234 \\ + \quad \underline{1234} \\ 1370974 \end{array}$$

1961 last name

Karatsuba's algorithm

Consider the product of two integers

$$(x_1 \cdot 10^{n/2} + x_0) \cdot (y_1 \cdot 10^{n/2} + y_0) \quad \text{hashed}$$

$$x_0 y_1 + x_1 y_0 = (x_0 + y_1) \cdot (x_1 + y_0) - x_0 y_0 - x_1 y_1$$

$$a \cdot b = x_1 y_1 10^n + [\quad] \cdot 10^{\frac{n}{2}} + x_0 y_0$$

This has 3 multiplications..

$$T(n) = 3 \cdot T\left(\frac{n}{2}\right) + O(n), \quad T(n) = \Theta\left(n^{\log 3}\right)$$

$$T(n) = \Theta\left(n^{1.58}\right)$$

Discussion Problem 4

FFT
$O(n \log n)$

Consider another divide and conquer algorithm for integer multiplication. 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. What would be the runtime complexity of this multiplication?

154|517|766

$\frac{n}{3}$

$a = x_2 \cdot 10^{2n/3} + x_1 \cdot 10^{n/3} + x_0$

$b = y_2 10^{2n/3} + y_1 10^{n/3} + y_0$

Recur. equation: $T(n) = \boxed{9} \cdot T\left(\frac{n}{3}\right) + O(n)$

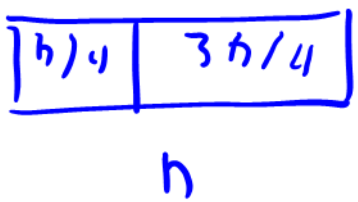
1962, Cook from MIT
he reduced 9 to 5
multiplications.

$T(n) = \Theta(n^2)$

$T(n) = \Theta(n^{\log_3 5})$ better than Karatsuba's

Discussion Problem 5

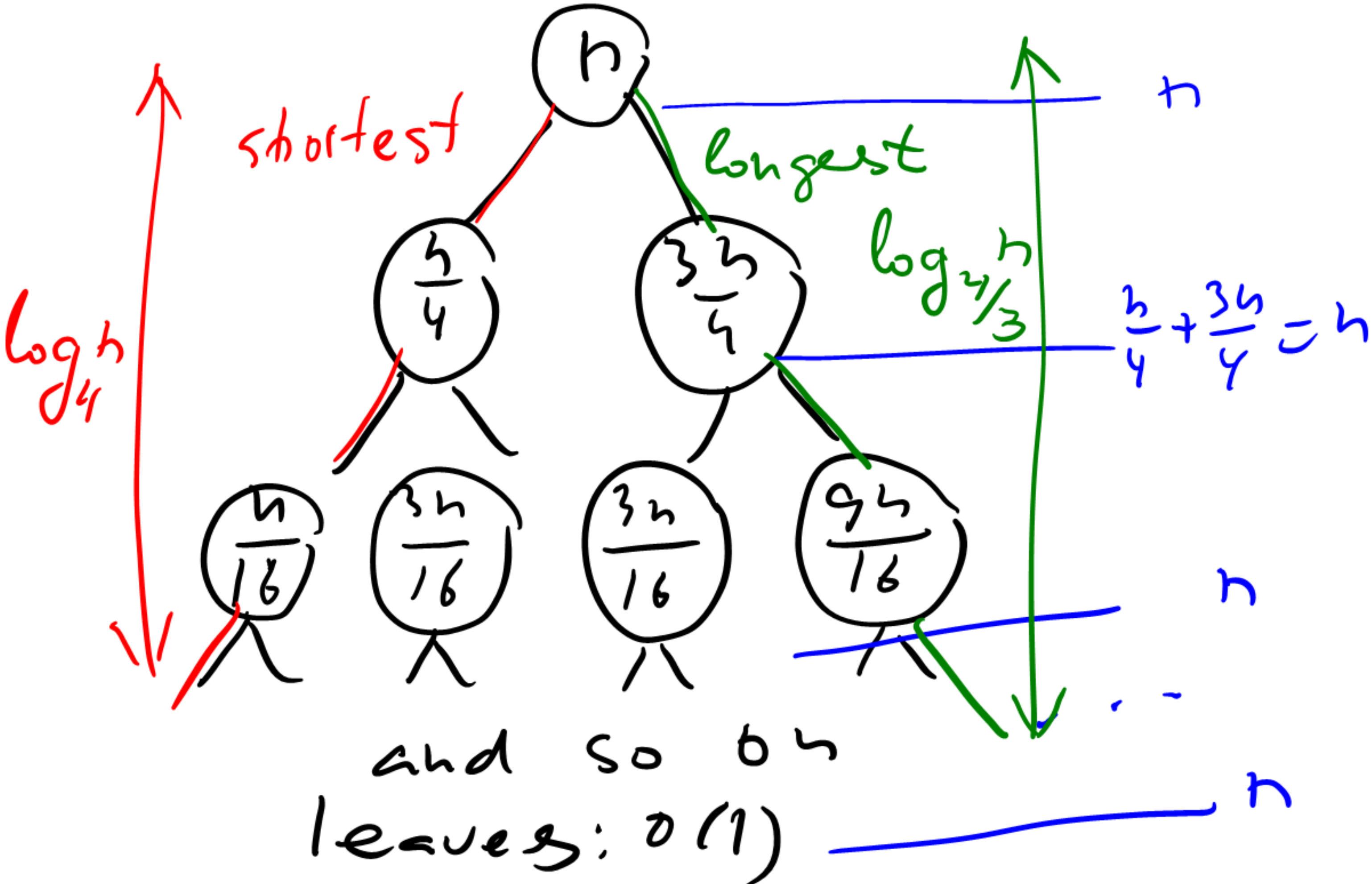
Design a new Mergesort algorithm in which instead of splitting the input array in half we split it in the ratio 1:3. What is the runtime complexity of this algorithm?



cannot apply the MT

Recurrence: $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + O(n)$

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



$T(n) \approx n \cdot \text{height}$

$$\left.\begin{aligned} n \cdot \log_4 n &\leq T(n) \leq n \cdot \log_{\frac{4}{3}} n \\ c_2 \cdot n \cdot \log n &\leq T(n) \leq c_1 \cdot n \cdot \log n \end{aligned}\right\} \; T(n) = \Theta(n \log n)$$

Discussion Problem 6

There are 2 sorted arrays A and B of size n each. Design a D&C algorithm to find the median of the array obtained after merging the above 2 arrays (i.e. array of length 2n). Discuss its runtime complexity

Input: A and B

O(1)

A = [~~1, 3, 5~~, 16, 18, 21, 30]

B = [2, 13, 17, 20, ~~23, 29, 35~~]

AUB = [1,2,3,5,13,16,17,18,20,21,23,29,30,35]

1. No D&C: O(n) by merging 2 sorted arrays.
2. D&C: $\boxed{O(\log n)}$ clue!! binary search

$A' = [16, 18, 21, \cancel{30}]$ Input size: $n/2$

$B' = [\cancel{2}, 13, 17, 20]$

$A'' = [16, 18, 21]$ Input size $n/4$

$B'' = [13, 17, 20]$

and so on

Recurrence: $T(n) = 1 \cdot T\left(\frac{n}{2}\right) + O(1)$

same as binary search

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

CPU for integer multiplication

GPU $\longrightarrow$ # Matrix Multiplication

NVIDIA

TPU tensor

Google

gaming, ML, crypto, ...

A B $C = A \times B$, size $n \times n$

$= n^2$

$O(n)$

$$\begin{array}{cc} & n \\ n & \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \end{array} \begin{array}{c} n \\ \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \end{array} = \begin{array}{cc} & \\ n & \begin{pmatrix} \boxed{a_{11}b_{11} + a_{12}b_{21}} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix} \end{array}$$

Runtime: $O(n^2 \cdot n) = O(n^3)$

$(a_{11}, a_{12}) \cdot (b_{11}, b_{21}) \in O(n)$

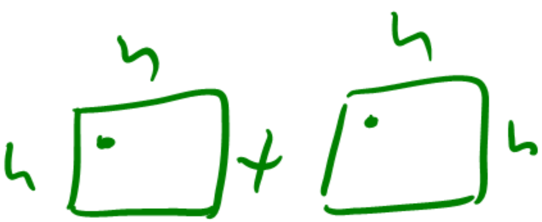
Matrix Multiplication

The usual rules of matrix multiplication holds for block matrices

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} \overset{1}{A_{11}B_{11}} + \overset{2}{A_{12}B_{21}} & \overset{3}{A_{11}B_{12}} + \overset{4}{A_{12}B_{22}} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

addition

$O(n^2)$



Algorithm

Let $n = 2^k$ and $M(A,B)$ denote the matrix product

1. if A is 1x1 matrix, return $a_{11} * b_{11}$.
2. write $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$

   where A_{ij} and B_{ij} are n/2 x n/2 matrices.
3. Compute $C_{ij} = M(A_{i1},B_{1j}) + M(A_{i2},B_{2j})$
4. Return $\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

$T(n) = \boxed{8}\,T\left(\frac{n}{2}\right) + O(n^2)$, $T(n) = \Theta(n^3)$

1968,

Strassen's Algorithm

How many additions? 18

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} s_1+s_2-s_4+s_6 & s_4-s_5 \\ \boxed{s_6+s_7} & s_2-s_3+s_5-s_7 \end{pmatrix}$$

$a_{21}b_{11}+a_{22}b_{21}$

$s_1 = (a_{12}-a_{22})\,(b_{21}+b_{22})$

$s_2 = (a_{11}+a_{22})\,(b_{11}+b_{22})$

$s_3 = (a_{11}-a_{21})\,(b_{11}+b_{12})$

$s_4 = (a_{11}+a_{12})\,b_{22}$

$s_5 = a_{11}\,(b_{12}-b_{22})$

$s_6 = a_{22}\,(b_{21}-b_{11})$

$s_7 = (a_{21}+a_{22})\,b_{11}$

It takes 7 multiplications

$T(n) = 7 \cdot T\left(\frac{n}{2}\right) + O(n^2)$

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

$T(n) = \Theta(n^{2.8})$

$s_6+s_7 = a_{22}(b_{21}-b_{11}) + b_{11}(a_{21}+a_{22}) =$

$= a_{22}b_{21} - \cancel{a_{22}b_{11}} + b_{11}a_{21} + \cancel{b_{11}a_{22}}$

Fast Matrix Multiplication

1969, Strassen $O(n^{2.808})$.
1978, Pan $O(n^{2.796})$
1979, Bini $O(n^{2.78})$
1981, Schonhage $O(n^{2.548})$
1981, Pan $O(n^{2.522})$
1982, Romani $O(n^{2.517})$
1982, Coppersmith and Winograd $O(n^{2.496})$, NSA
1986, Strassen $O(n^{2.479})$ library
1989, Coppersmith and Winograd $O(n^{2.376})$
2010, Stothers $O(n^{2.374})$
2011, Williams $O(n^{2.3728642})$
2014, Le Gall $O(n^{2.3728639})$ theoretical~!!

Discussion Problem 7

You are given an $\boxed{\text{unsorted}}$ array of ALL integers in the range $[0, \ldots, 2^k-1]$ except for one integer, denoted the missing number by M.

Describe a <u>divide-and-conquer</u> to find the missing number *M*, and discuss its the worst-case runtime complexity in terms of $n = 2^k$.

$k = 3$ $[0, 1, 2, 3, \cancel{4}, 5, 6, 7]$

Input: array of integers

Goal: split the input

$A = [A_1, A_2]$

$M \in A_1$?

$M \in A_2$?

input: $[5, 6, 7, 1, 2, 3, 5]$

What number is missed?

size n

~~100~~
000
~~111~~
001
011
~~101~~
~~110~~

size $\frac{n}{2}$

000
001
011

$M = 010$

missing number starts with 0
followed by 1

Runtime: $T(n) = 1 \cdot T(\frac{n}{2}) + \underbrace{O(n)}_{\text{traverse the input}}$

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

Finding the Maximum Subsequence Sum

Given an array A[0,…, n-1] of integers, design a D&C algorithm that finds a subarray A[i, …, j] such that A[i] + A[i + 1] + … + A[j] is the maximum.

A_1 A_2

For example,

$$A = \{3, -4, 5, -2, -2, 6, -3, 5, -3, 2\}$$

Output: {5, -2, -2, 6, -3, 5}

Sum = 5-2-2+6-3+5 = 9

Finding the Maximum Subsequence Sum (MSS)

3, -4, 5, -2, -2, 6, -3, 5, -3, 2

A_1 A_2

$(l_1, r_1, max_1) = MSS(A_1)$; recursive

$(l_2, r_2, max_2) = MSS(A_2)$; recursive

$(l_3, r_3, max_3) = span(A_1 \cup A_2)$; iterative

return $MAX(max_1, max_2, max_3)$;

Finding the Maximum Subsequence Sum (MSS)

A_1 A_2

3, -4, 5, -2, -2, 6, -3, 5, -3, 2

Implementation of span?

-2 must be a part of span

6 must be a part of span

Compute partial sums

0, -3, [1, -4, -2 | 6, 3, 8,] 5, 7

ℓ_3 r_3

Find the max sequence, $max_3 = 9$

Runtime: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \underbrace{O(n)}_{\text{span}}$