Design and Analysis of Algorithms I

Introduction

Guiding Principles

Guiding Principle #1

"worst – case analysis" : our running time bound holds for every input of length n.
-Particularly appropriate for "general-purpose" routines

As Opposed to
--"average-case" analysis
--benchmarks

REQUIRES DOMAIN KNOWLEDGE

BONUS : worst case usually easier to analyze.

Guiding Principle #2

Won't pay much attention to constant factors, lower-order terms

<u>Justifications</u>

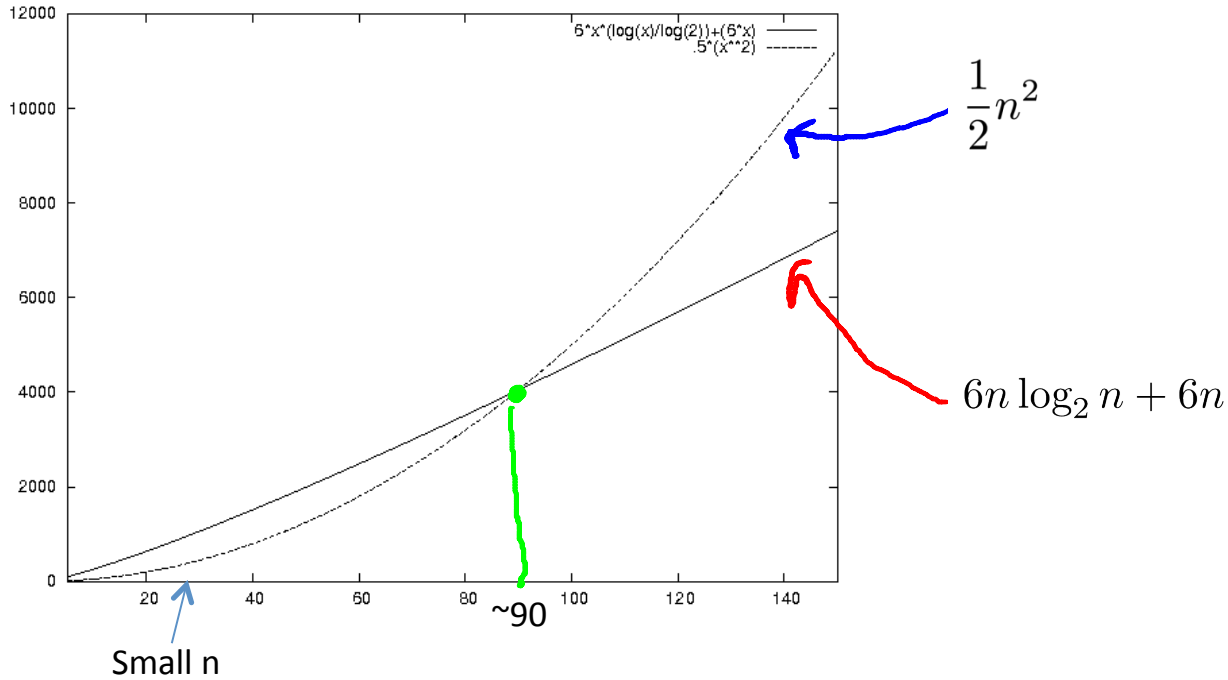
1. Way easier
2. Constants depend on architecture / compiler / programmer anyways
3. Lose very little predictive power (as we'll see)

Guiding Principle #3

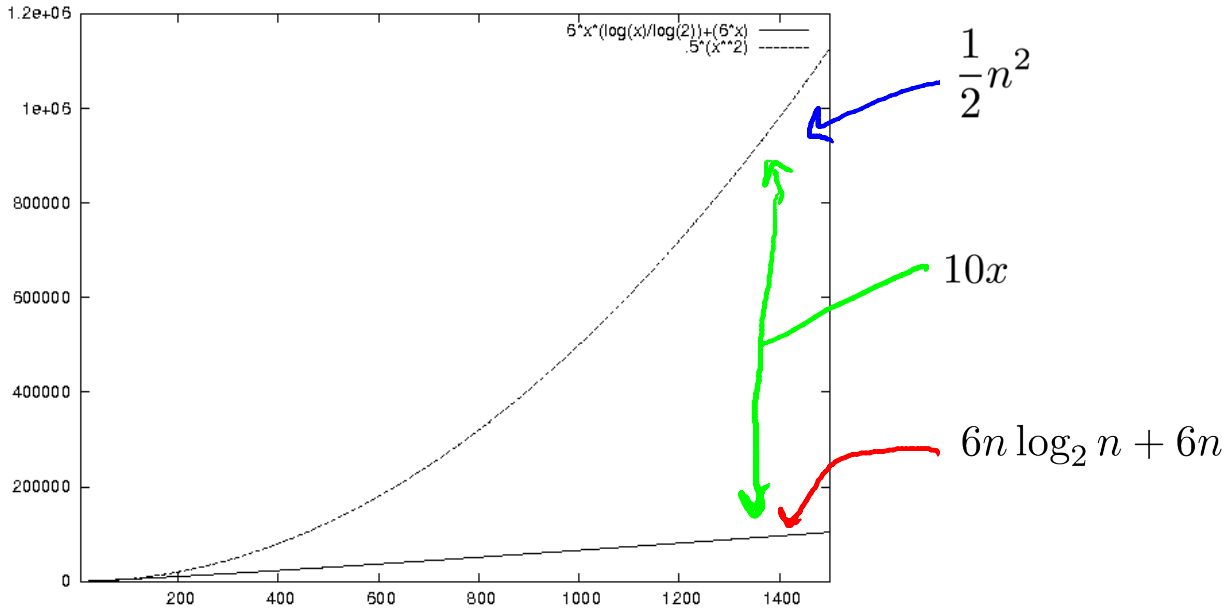
<u>Asymptotic Analysis</u> : focus on running time for large input sizes n

Eg : $\underbrace{6n\log_2 + 6n}_{\text{MERGE SORT}}$ "better than" $\underbrace{\frac{1}{2}n^2}_{\text{INSERTION SORT}}$

<u>Justification:</u> Only big problems are interesting!



Tim Roughgarden



Tim Roughgarden

What Is a "Fast" Algorithm?

<u>This Course</u> : adopt these three biases as guiding principles

fast algorithm $\approx$ worst-case running time grows slowly with input size

<u>Usually</u> : want as close to linear (O(n)) as possible

Design and Analysis of Algorithms I

Introduction

Integer Multiplication

Integer Multiplication

Input : 2 n-digit numbers x and y

Output : product x*y

“Primitive Operation” - add or multiply 2 single-digit numbers

The Grade-School Algorithm

```
      5678
    x 1234
  ---------
     22712
    17034-
   11356--
   5678---
  ---------
   7006652
```

Roughly n operations per row up to a constant

# of operations overall ~ constant* n^2

The Algorithm Designer's Mantra

"Perhaps the most important principle for the good algorithm designer is to refuse to be content."

-Aho, Hopcroft, and Ullman, *The Design and Analysis of Computer Algorithms*, 1974

CAN WE DO BETTER ?
[than the "obvious" method]

Design and Analysis of Algorithms I

Introduction

Karatsuba Multiplication

Example

A Recursive Algorithm

Write $x = 10^{n/2}a + b$ and $y = 10^{n/2}c + d$

Where a,b,c,d are n/2-digit numbers.

[example: a=56, b=78, c=12, d=34]

Then $x.y = (10^{n/2}a + b)(10^{n/2}c + d)$

$$= (10^n ac + 10^{n/2}(ad + bc) + bd \qquad (*)$$

Idea : recursively compute ac, ad, bc, bd, then compute (*) in the obvious way

Simple Base Case Omitted

Karatsuba Multiplication

$$x.y = (10^n ac + 10^{n/2}(ad + bc) + bd$$

1. Recursively compute ac
2. Recursively compute bd
3. Recursively compute (a+b)(c+d) = ac+bd+ad+bc

Gauss' Trick : (3) – (1) – (2) = ad + bc

Upshot : Only need 3 recursive multiplications (and some additions)

Q : which is the fastest algorithm ?

Design and Analysis of Algorithms I

Introduction

Merge Sort (Overview)

Why Study Merge Sort?

- Good introduction to divide & conquer
  - Improves over Selection, Insertion, Bubble sorts
- Calibrate your preparation
- Motivates guiding principles for algorithm analysis (worst-case and asymptotic analysis)
- Analysis generalizes to "Master Method"

The Sorting Problem

Input : array of n numbers, unsorted.

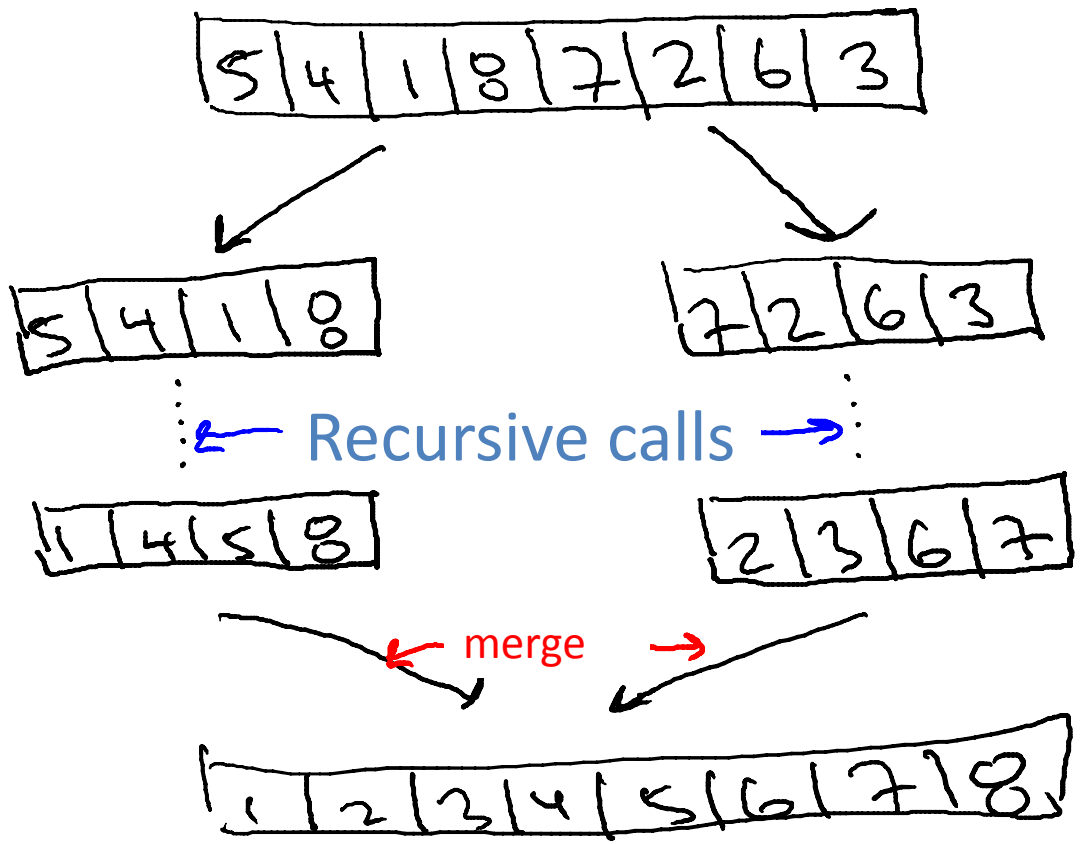
5	4	1	8	7	2	6	3

<u>Assume Distinct</u>

Output : Same numbers, sorted in increasing order

1	2	3	4	5	6	7	8

Merge Sort: Example



Tim Roughgarden

Design and Analysis of Algorithms I

Introduction

Merge Sort (Pseudocode)

Tim

Merge Sort: Pseudocode

-- recursively sort 1st half of the input array

-- recursively sort 2nd half of the input array

-- merge two sorted sublists into one

[ignores base cases]

Pseudocode for Merge:

```
C = output [length = n]
A = 1st sorted array [n/2]
B = 2nd sorted array [n/2]
i = 1
j = 1
```

```
for k = 1 to n
        if A(i) < B(j)
                C(k) = A(i)
                i++
        else [B(j) < A(i)]
                C(k) = B(j)
                j++
end
```

(ignores end cases)

Merge Sort Running Time?

Key Question : running time of Merge Sort on array of n numbers ?

[running time ~ # of lines of code executed]

Pseudocode for Merge:

C = output [length = n]

A = 1st sorted array [n/2]

B = 2nd sorted array [n/2]

i = 1

j = 1

2 operations

```
for k = 1 to n
        if A(i) < B(j)
                C(k) = A(i)
                i++
        else [B(j) < A(i)]
                C(k) = B(j)
                j++
end
```

(ignores end cases)

Tim Roughgarden

Running Time of Merge

Upshot : running time of Merge on array of m numbers is $\leq 4m + 2$

$\leq 6m$ (Since $m \geq 1$)

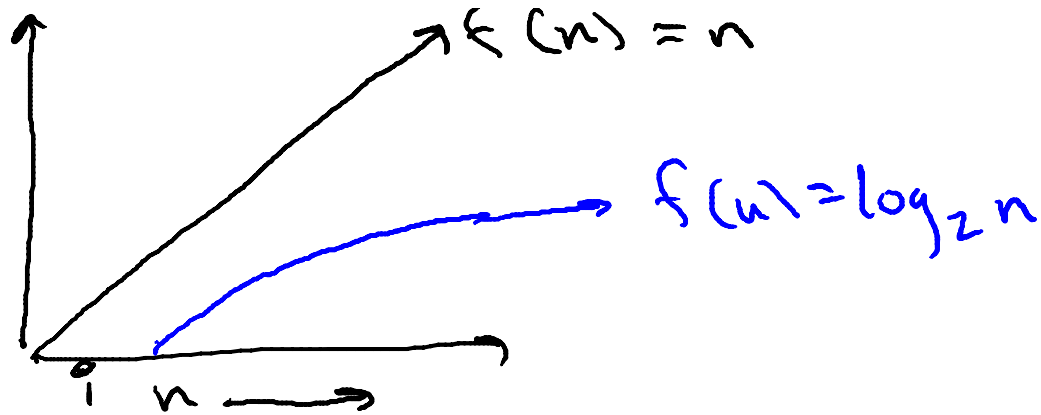
Running Time of Merge Sort

Claim : Merge Sort requires

$\leq 6n \log_2 n + 6n$ operations

to sort n numbers.

Recall : = $\log_2 n$ is the # of times you divide by 2 until you get down to 1



Design and Analysis of Algorithms I

Introduction

Merge Sort (Analysis)

Tim

Running Time of Merge Sort

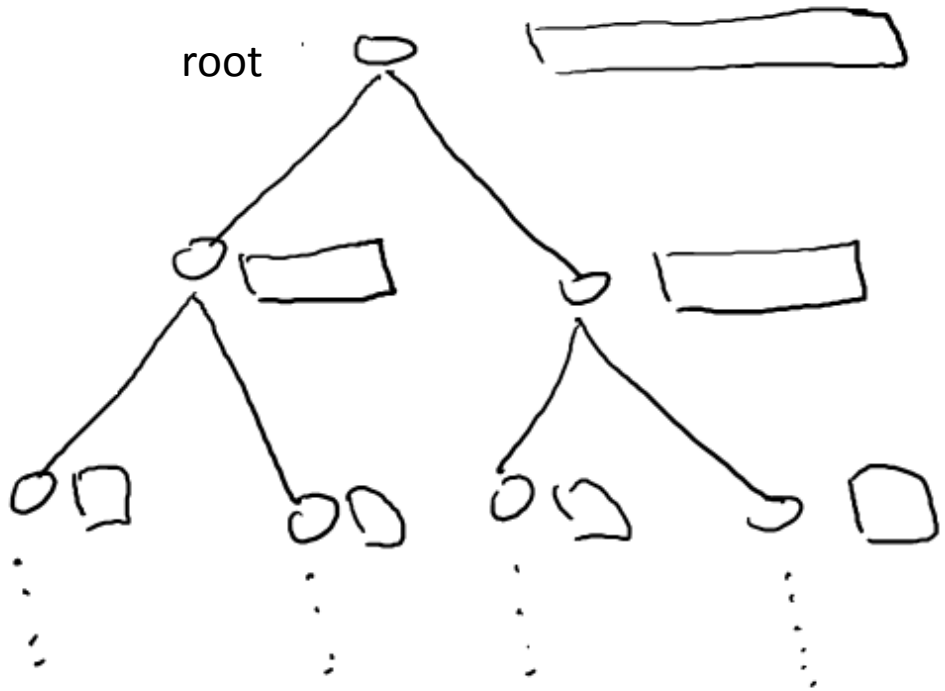
Claim: For every input array of n numbers, Merge Sort produces a sorted output array and uses at most $6n \log_2 n + 6n$ operations.

Proof of claim (assuming n = power of 2):

Level 0
[outer call to
Merge Sort]

Level 1
(1st recursive
calls)

Level 2

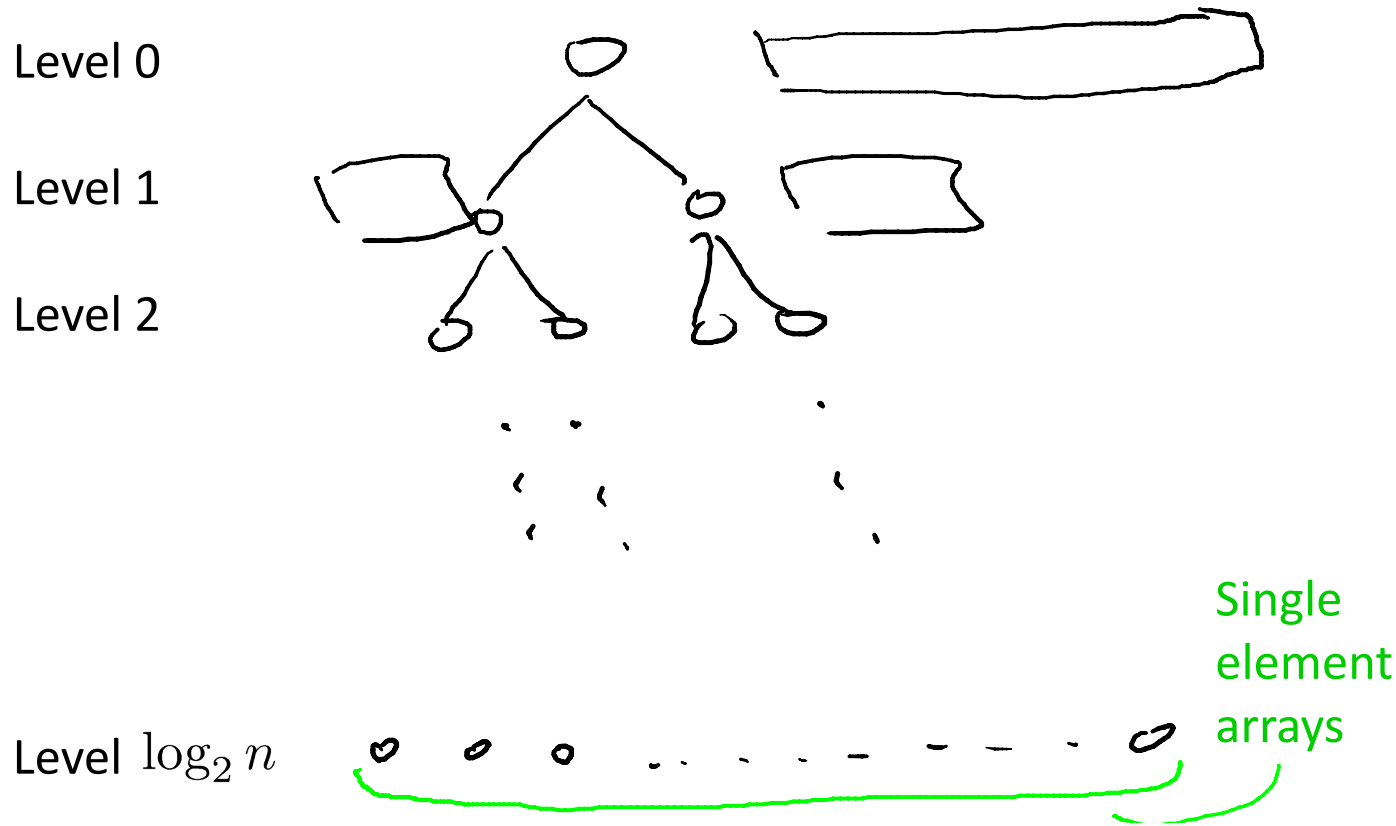


Ter
ver

Roughly how many levels does this recursion tree have (as a function of n, the length of the input array)?

- A constant number (independent of n).
- $\log_2 n$ $(\log_2 n + 1)$ to be exact!
- $\sqrt{n}$
- n

Proof of claim (assuming n = power of 2):



Ter
ver

What is the pattern ? Fill in the blanks in the following statement: at each level j = 0,1,2,.., $\log_2 n$, there are <blank> subproblems, each of size <blank>.

- 2^j *and* 2^j, *respectively*
- $n/2^j$ *and* $n/2^j$, *respectively*
- 2^j *and* $n/2^j$, *respectively*
- $n/2^j$ *and* 2^j, *respectively*

Proof of claim (assuming n = power of 2) :

At each level j=0,1,2,.., $\log_2 n$,

Total # of operations at level j = 0,1,2,...,$\log_2 n$

$$\leq 2^j * 6(\frac{n}{2^j}) = 6n$$

# of level-j subproblems

Size of level-j subproblem

Work per level – j subproblem

$\boxed{\text{Total}}$

$$6n(\log_2 n + 1)$$

Work per level

# of levels

Tim Roughgarden

Running Time of Merge Sort

Claim: For every input array of n numbers, Merge Sort produces a sorted output array and uses at most $6n \log_2 n + 6n$ operations.

QED!