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"

Tim Roughgarden

The Sorting Problem

Input : array of n numbers, unsorted.

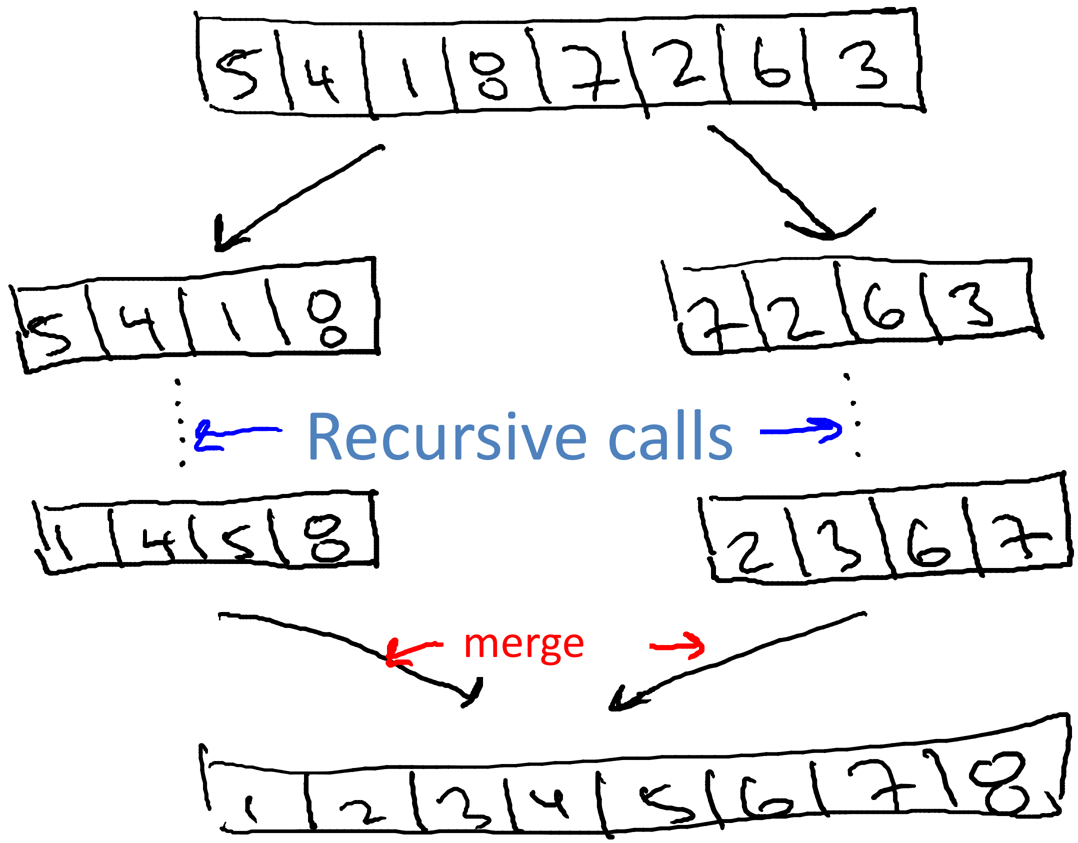
5	4	1	8	7	2	6	3

<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]

Tim Roughgarden

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)

Tim Roughgarden

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

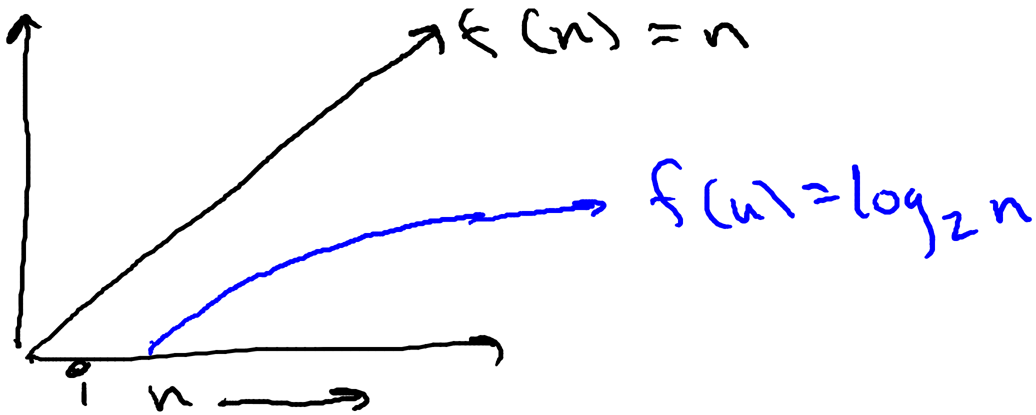
Upshot : running time of Merge on array of m numbers is $\leq 4m + 2$

$\leq 6m$ (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



Tim Roughgarden

Design and Analysis of Algorithms I

Introduction

Merge Sort (Analysis)

Tim

Running Time of Merge Sort

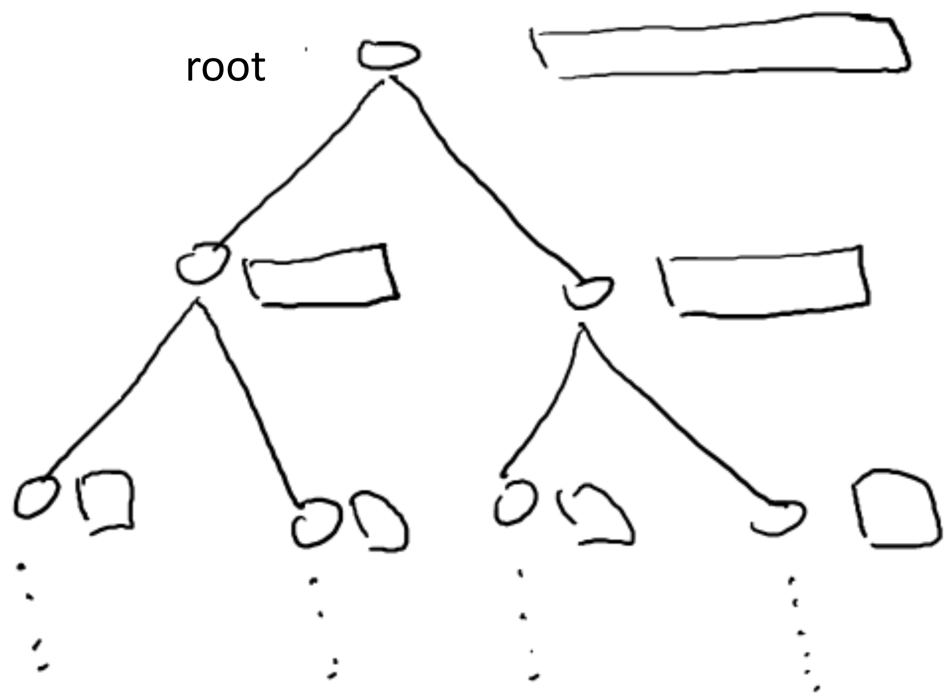
Claim: For every input array of n numbers, Merge Sort produces a sorted output array and uses at most $6n \log_2 n + 6n$ operations.

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

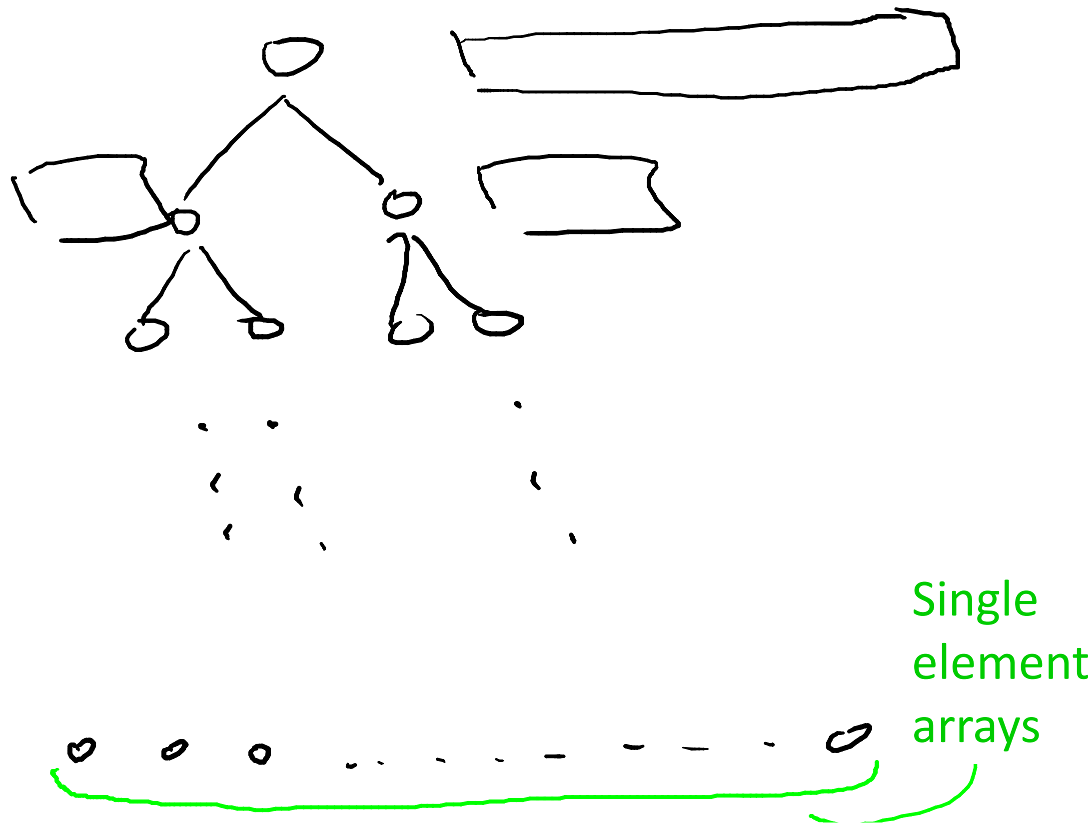
Proof of claim (assuming n = power of 2):

Level 0

Level 1

Level 2

Level $\log_2 n$



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,…,$\log_2 n$

$$\leq \underbrace{2^j}_{\text{\# of level-j subproblems}} * \underbrace{6(\tfrac{n}{\underbrace{2^j}_{\text{Size of level-j subproblem}}})}_{\text{Work per level – j subproblem}} = 6n$$

# of level-j subproblems

Size of level-j subproblem

Work per level – j subproblem

Total

$$\underbrace{6n}_{\text{Work per level}}(\log_2 \underbrace{n + 1}_{\text{\# of levels}})$$

Work per level

# of levels

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!