Lecture 06: Sorting by Divide and Conquer II

COSC 311 *Algorithms*, Fall 2022

Announcement

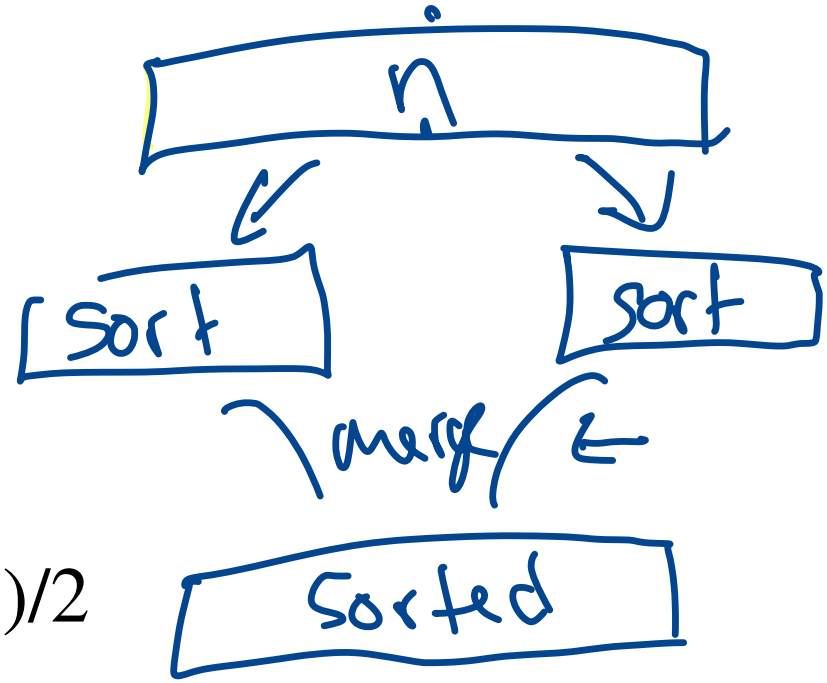
Accountability Groups

Overview

1. MergeSort
2. Running Time of Merge Sort
3. QuickSort

Previously

- Sorting meets Divide & Conquer
- MergeSort: Divide by Index



1. divide a into halves $m = (n + 1)/2$
2. sort $a[1..m - 1]$ recursively
3. sort $a[m..n]$ recursively
4. merge $a[1..m - 1]$ and $a[m..n]$ to form sorted array

Pseudocode

```
# sort values of a between indices i and j-1
MergeSort(a, i, j):
  if j - i = 1 then
    return
  endif
  m <- (i + j) / 2
  MergeSort(a,i,m)
  MergeSort(a,m,j)
  Merge(a,i,m,j)
```

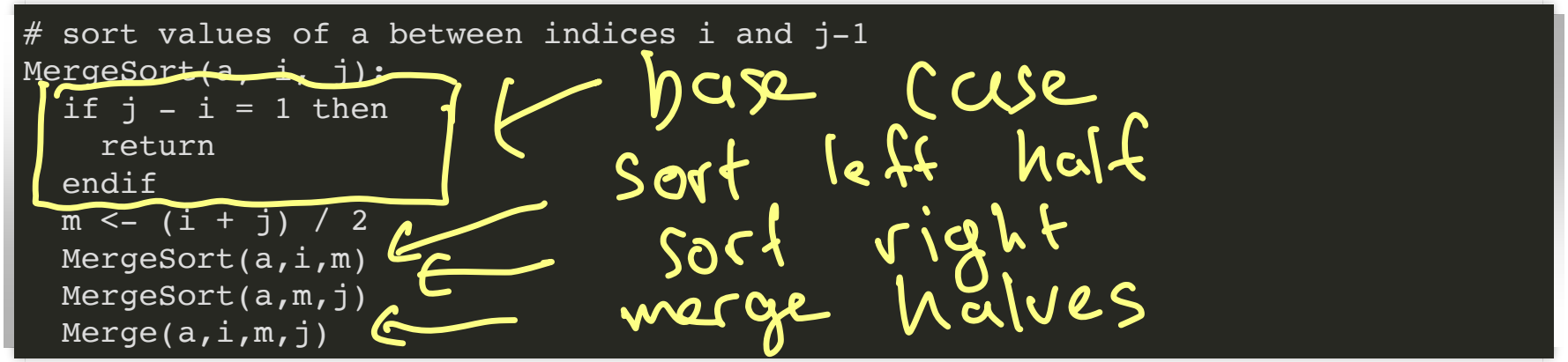
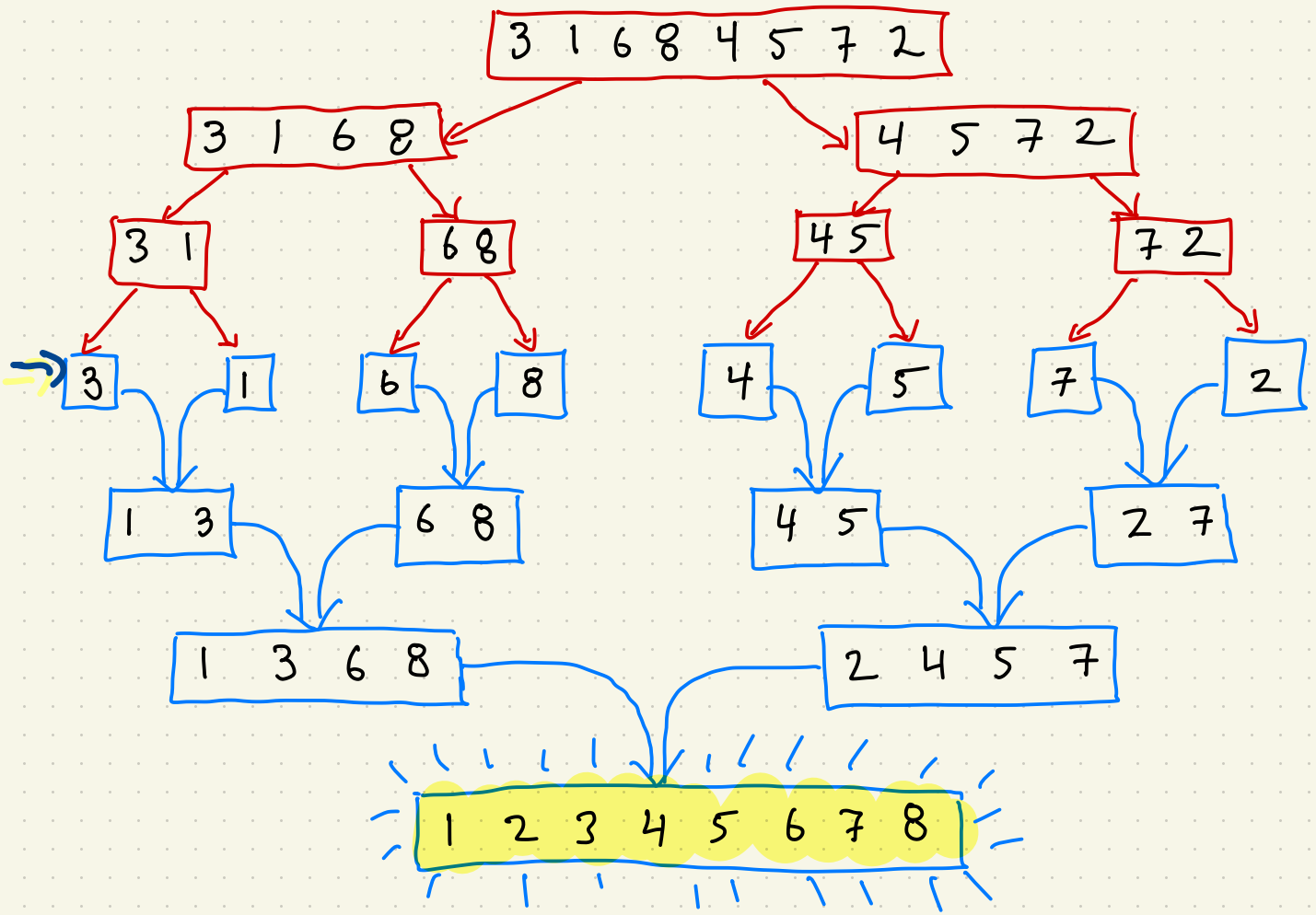
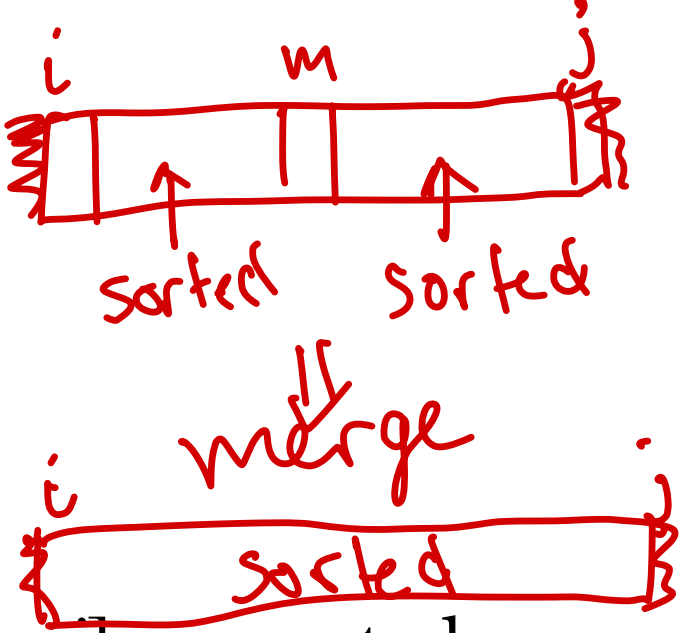


Illustration of MergeSort


3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
4 5
7 2
3
1
6
8
4
5
7
2
1 3
6 8
4 5
2 7
1 3 6 8
2 4 5 7
1 2 3 4 5 6 7 8



Correctness of MergeSort

Establish two claims:

Claim 1 (merge). If $a[i..m-1]$ and $a[m..j]$ are sorted, then after Merge(a, i, m, j), $a[i..j]$ is sorted.

- Argued on lecture ticket!

Claim 2. For any indices $i < j$, after calling MergeSort(a, i, j), $a[i..j]$ is sorted.

- Argue by Induction!

Pseudocode Again

```
00  # sort values of a between indices i and j-1
01  MergeSort(a, i, j):
02    if j - i = 1 then
03      return
04    endif
05    m <- (i + j) / 2
06    MergeSort(a,i,m)
07    MergeSort(a,m,j)
08    Merge(a,i,m,j)
```

k=1 just return
all arrays of size 1
are sorted!

have size $\frac{k+1}{2} < k$

Output is sorted by claim 1

Succeed by ind. hyp.

Inductive Claim

of elements being sorted

Consider MergeSort(a, i, j), define $k = j - i$ to be *size*

$P(k)$: for every $k' \leq k$, MergeSort(a, i, j) with size k' succeeds

Base case $k = 1$:

all arrays of size 1 are sorted!

Inductive step $P(k) \implies P(k+1)$:

Assume Mergesort sorts all arrays of size up to k.

Show sorts arrays of size k+1:

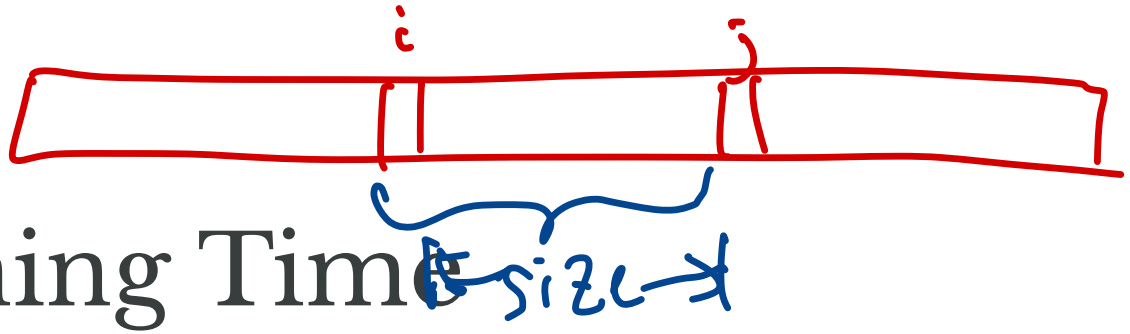
Selection sort: $O(n^2)$

Question

How efficient is MergeSort?

```
00  # sort values of a between indices i and j-1
01  MergeSort(a, i, j):
02    if j - i = 1 then
03      return
04    endif
05    m <- (i + j) / 2
06    MergeSort(a,i,m)
07    MergeSort(a,m,j)
08    Merge(a,i,m,j)
```

$O(n \log n)$



Analyzing Running Time

```
00  # sort values of a between indices i and j-1
01  MergeSort(a, i, j):
02    if j - i = 1 then
03      return
04    endif
05    m <- (i + j) / 2
06    MergeSort(a,i,m)
07    MergeSort(a,m,j)
08    Merge(a,i,m,j)
```

O(1) ?? O(k)

Observation 1. Let $k = j - i$ be the **size** of the method call MergeSort(a, i, j). Then running time is $O(k)$+ running time of recursive calls on lines 6-7.

Observation 2. Recursive calls have size $k/2$.

- Assume size is power of 2

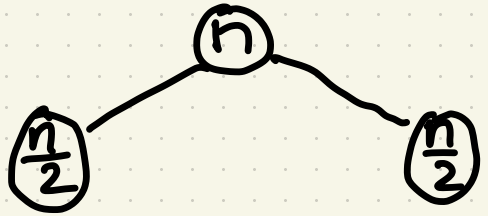
Combining Observations

Looking @ recursive calls by depth:

Calls (size)	no. calls	size of calls
(n)	1	n

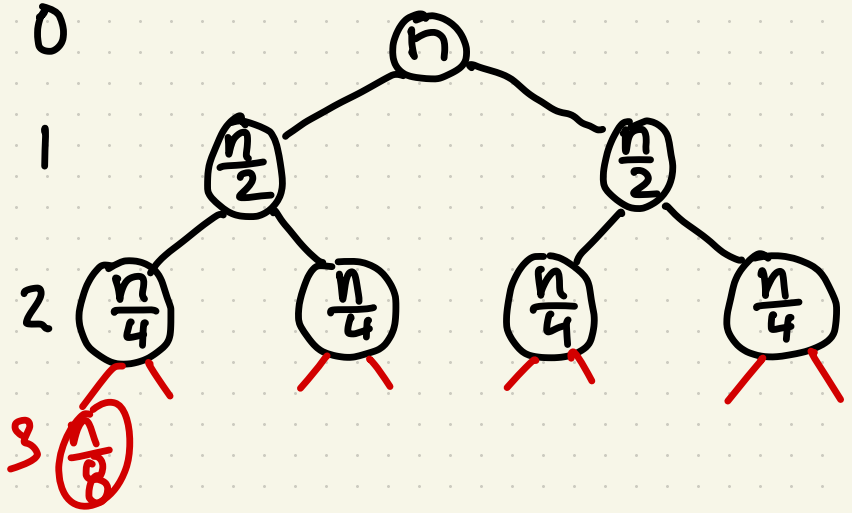
Looking @ recursive calls by <u>depth</u>:

Calls (size)



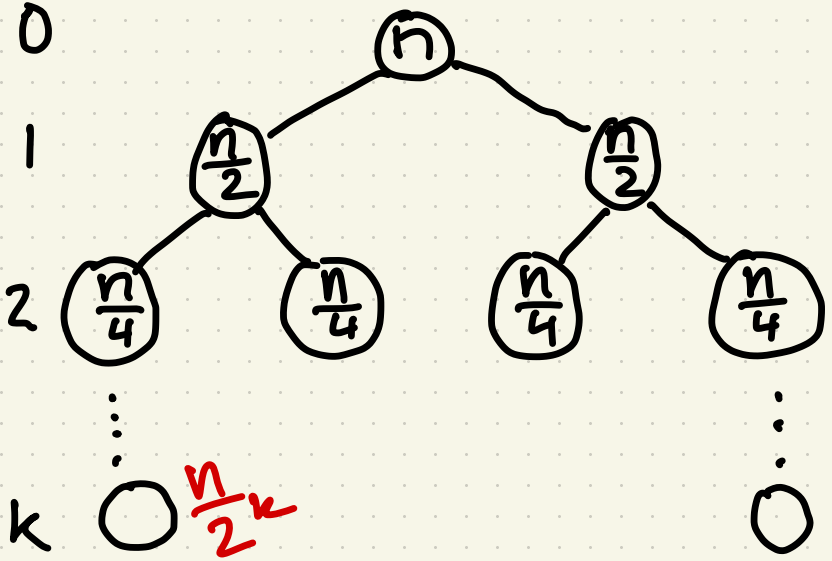
no. calls	size of calls
1	n
2	$\frac{n}{2}$

Looking @ recursive calls by depth:



	no. calls	size of calls
0	1	n
1	2	$\frac{n}{2}$
2	4	$\frac{n}{4}$
3	8	$\frac{n}{8}$

Looking @ recursive calls by depth:



no. calls

1

2

4

2^k

size of calls

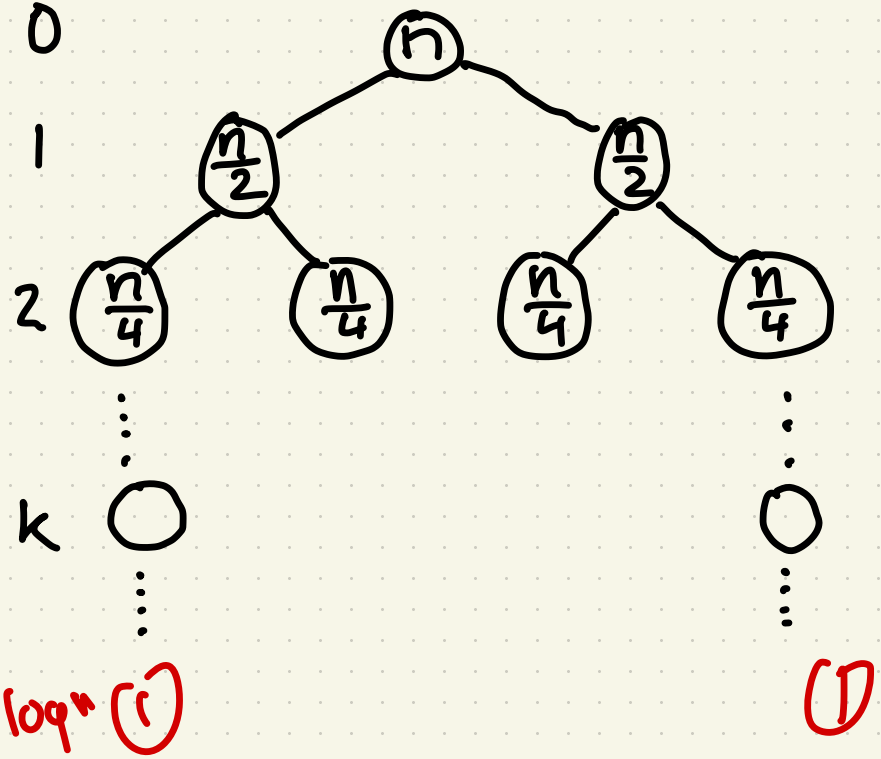
n

$\frac{n}{2}$

$\frac{n}{4}$

$\frac{n}{2^k}$

Looking @ recursive calls by depth:



	no. calls	size of calls
0	1	n
1	2	$\frac{n}{2}$
2	4	$\frac{n}{4}$

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log n$$

When do we stop?

Recall Logarithms (base 2)

Define log by

- $\log a = b \iff 2^b = a$

Another way

- $\log a$ is # times a can be divided by 2 to get (at most) 1.

Facts.

1. For every constant $c > 0$, $\log n = O(n^c)$.
2. $\log n \neq O(1)$.

Running Times by Depth

merge, etc.

depth	no.	size	time (not counting rec. calls)
0	1	n	$1 \cdot O(n) = O(n)$
1	2	$n/2$	$2 \cdot O(n/2) = O(n)$
2	4	$n/4$	$4 \cdot O(n/4) = O(n)$
⋮	⋮	⋮	⋮
k	2^k	$n/2^k$	$2^k O(\frac{n}{2^k}) = O(n)$
⋮	⋮	⋮	
$\log n$	n	1	$n\,O(1) = O(n)$

$\log n$ depth

$\Rightarrow$ total r.t. $(\log n) \cdot O(n)$

$O(n \log n)$

Foreshadowing

$T(n)$ = running time of MergeSort on a of size n

worst-case

$T(n)$ satisfies the recursion relation

merge, etc.

$$T(n) = 2 \cdot T(n/2) + \Theta(n)$$

recursive calls

"Master Method" gives gen. formula for this type of recursion

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

Picture so Far:

SelectionSort. $O(n^2)$ operations

- $O(n^2)$ comparisons
- $O(n)$ swaps

BubbleSort and **InsertionSort.** $O(n^2)$ operations

- $O(n^2)$ comparisons
- $O(n^2)$ swaps

MergeSort. $O(n \log n)$ operations

- $O(n \log n)$ comparisons
- $O(n \log n)$ modifications
- uses $O(n)$ space overhead