



"Organizing is what you do before you do something, so that when you do it, it is not all mixed up."

~ A. A. Milne

SORTING

Prelim 1

2

- Tonight!!!!
- Two Sessions:
 - ▣ You should know by now what room to take the final. Jenna emailed you.
- Bring your Cornell ID!!!
- We will grade this evening, and if everything works out well, you will receive an email in early morning from Gradescope telling you to look at your grade.

Prelim 1

3

- Recitation 5. next week:
Enums and Java Collections classes.
Nothing to prepare for it!

But get A3 done.

Why Sorting?

4

- Sorting is useful
 - ▣ Database indexing
 - ▣ Operations research
 - ▣ Compression
- There are lots of ways to sort
 - ▣ There isn't one right answer
 - ▣ You need to be able to figure out the options and decide which one is right for your application.
 - ▣ Today, we'll learn several different algorithms (and how to develop them)

Some Sorting Algorithms

5

- Insertion sort
- Selection sort
- Quick sort
- Merge sort

InsertionSort

6

pre: b

0	b.length
?	

post: b

0	b.length
sorted	

inv: b

0	i	b.length
sorted	?	

or: $b[0..i-1]$ is sorted

inv: b

0	i	b.length
processed	?	

or: $b[0..i-1]$ is processed

A loop that processes elements of an array in increasing order has this invariant --- just replace “sorted” by “processed”.

What to do in each iteration?

8

inv:

0		i		b.length
b	sorted		?	

e.g.

0		i		b.length			
b	2	5	5	5	7	3	?

Loop body (inv true before and after)

2	5	5	5	3	7	?
2	5	5	3	5	7	?
2	5	3	5	5	7	?
2	3	5	5	5	7	?

Push $b[i]$ to its sorted position in $b[0..i]$, then increase i

This will take time proportional to the number of swaps needed

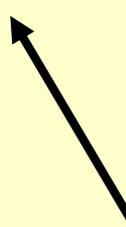
b	2	3	5	5	5	7	?
---	---	---	---	---	---	---	---

Insertion Sort

9

```
// sort b[], an array of int
// inv: b[0..i-1] is sorted
for (int i= 0; i < b.length; i= i+1) {
    // Push b[i] down to its sorted
    // position in b[0..i]
}
}
```

Present algorithm like this



Note English statement
in body.

Abstraction. Says **what**
to do, not **how**.

This is the best way to
present it. We expect
you to present it this
way when asked.

Later, can show how to
implement that with an
inner loop

Insertion Sort

10

```
// sort b[], an array of int
// inv: b[0..i-1] is sorted
for (int i= 0; i < b.length; i= i+1) {
    // Push b[i] down to its sorted
    // position in b[0..i]

    int k= i;
    while (k > 0 && b[k] < b[k-1]) {
        Swap b[k] and b[k-1];
        k= k-1;
    }
}
```

invariant P: $b[0..i]$ is sorted
except that $b[k]$ may be $< b[k-1]$

k					i	
2	5	3	5	5	7	?

example

start?

stop?

progress?

maintain

invariant?

Insertion Sort

11

```
// sort b[], an array of int
// inv: b[0..i-1] is sorted
for (int i= 0; i < b.length; i= i+1) {
    // Push b[i] down to its sorted
    // position in b[0..i]}
```

Pushing $b[i]$ down can take i swaps.

Worst case takes

$$1 + 2 + 3 + \dots + n-1 = (n-1)*n/2$$

swaps.

Let $n = b.length$

- Worst-case: $O(n^2)$
(reverse-sorted input)
- Best-case: $O(n)$
(sorted input)
- Expected case: $O(n^2)$

Performance

12

Algorithm	Ave time.	Worst-case time	Space	Stable?
Insertion Sort	$O(n^2)$.	$O(n^2)$	$O(1)$	Yes
Merge Sort				
Quick Sort				

We'll talk about stability later

SelectionSort

13

pre: b

0	b.length
?	

 post: b

0	b.length
sorted	

inv: b

0	i	b.length
sorted, $\leq b[i..]$	$\geq b[0..i-1]$	

 Additional term
in invariant

Keep invariant true while making progress?

e.g.: b

0	i	b.length										
1	2	3	4	5	6	9	9	9	7	8	6	9

Increasing i by 1 keeps inv true only if $b[i]$ is min of $b[i..]$

SelectionSort

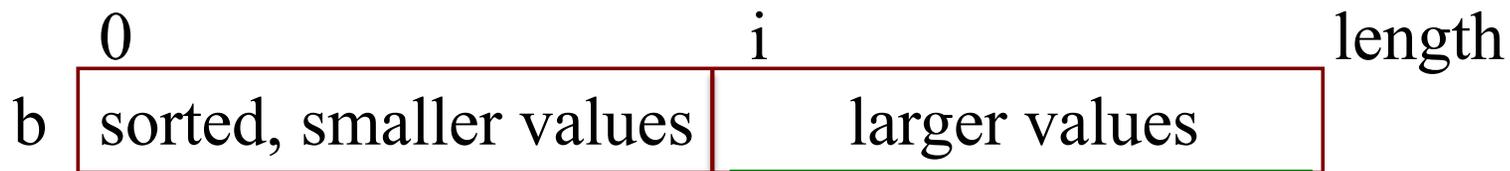
```
//sort b[], an array of int
// inv: b[0..i-1] sorted AND
//      b[0..i-1] <= b[i..]
for (int i= 0; i < b.length; i= i+1) {
    int m= index of min of b[i..];
    Swap b[i] and b[m];
}
```

Another common way for people to sort cards

Runtime

with $n = b.length$

- Worst-case $O(n^2)$
- Best-case $O(n^2)$
- Expected-case $O(n^2)$



Each iteration, swap min value of this section into $b[i]$

Performance

15

Algorithm	Ave time.	Worst-case time	Space	Stable?
Insertion sort	$O(n^2)$.	$O(n^2)$	$O(1)$	Yes
Selection sort	$O(n^2)$.	$O(n^2)$	$O(1)$	No
Quick sort				
Merge sort				

QuickSort

16

Quicksort developed by Sir Tony Hoare (he was knighted by the Queen of England for his contributions to education and CS).

84 years old.

Developed Quicksort in 1958. But he could not explain it to his colleague, so he gave up on it.

Later, he saw a draft of the new language Algol 58 (which became Algol 60). It had recursive procedures. First time in a procedural programming language. “Ah!,” he said. “I know how to write it better now.” 15 minutes later, his colleague also understood it.

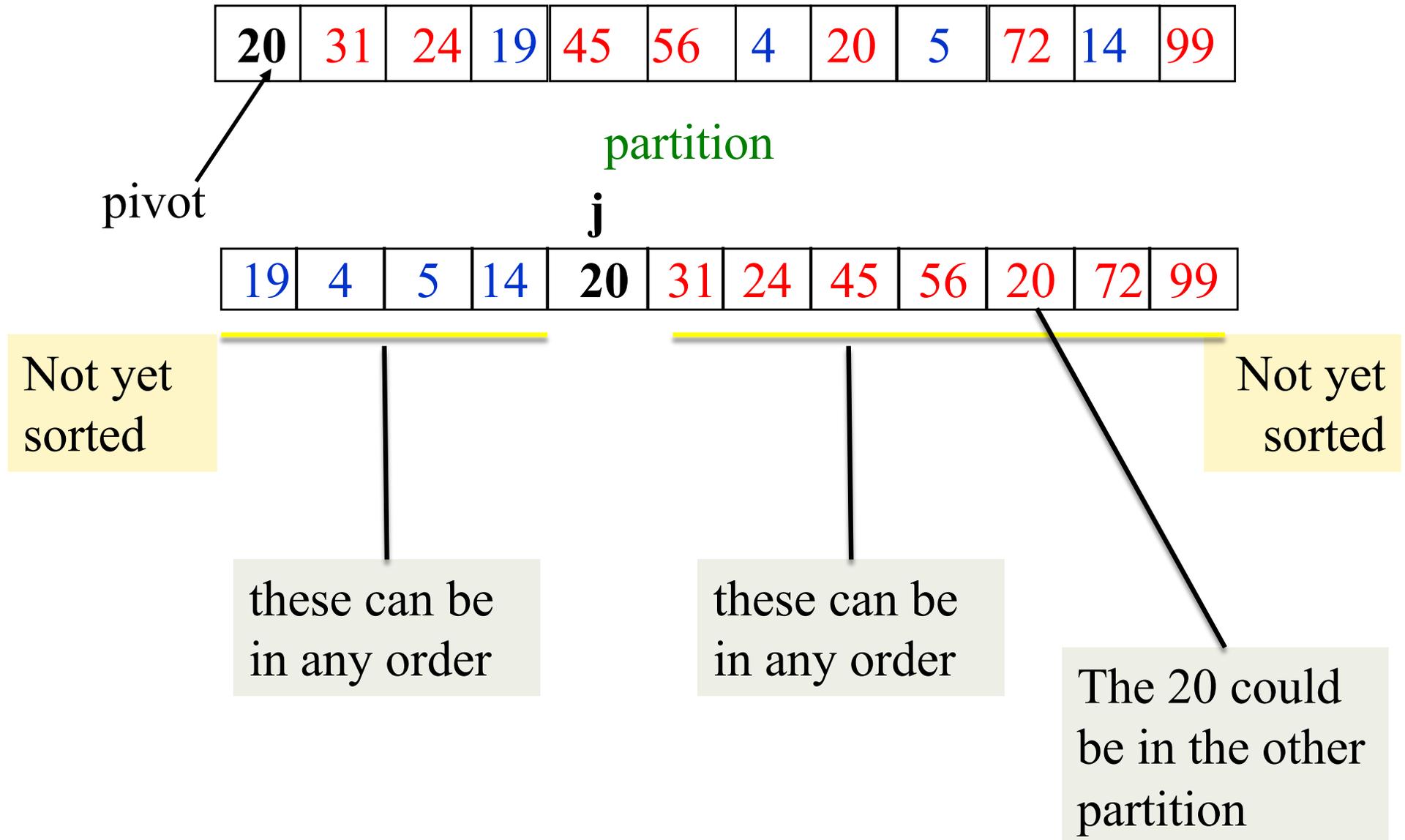


Dijkstras, Hoares, Grieses, 1980s

17



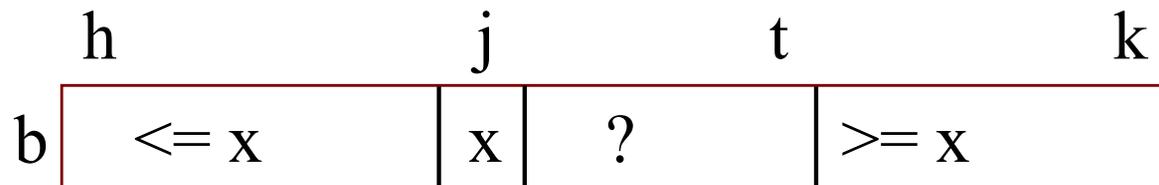
Partition algorithm of quicksort



Partition algorithm



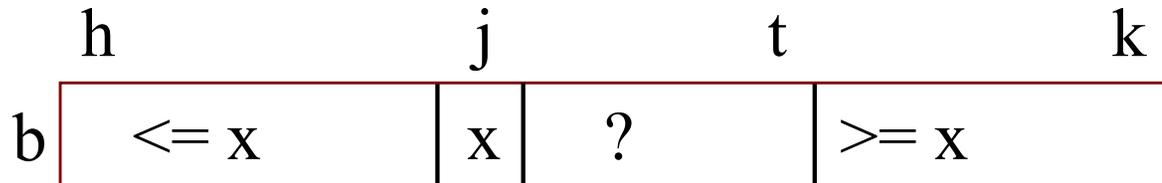
Combine pre and post to get an invariant



invariant
needs at
least 4
sections

Partition algorithm

21



```
j= h; t= k;
while (j < t) {
  if (b[j+1] <= b[j]) {
    Swap b[j+1] and b[j]; j= j+1;
  } else {
    Swap b[j+1] and b[t]; t= t-1;
  }
}
```

Takes linear time: $O(k+1-h)$

Initially, with $j = h$ and $t = k$, this diagram looks like the start diagram

Terminate when $j = t$, so the “?” segment is empty, so diagram looks like result diagram

QuickSort procedure

```
/** Sort b[h..k]. */
```

```
public static void QS(int[] b, int h, int k) {
```

```
    if (b[h..k] has < 2 elements) return; Base case
```

```
    int j= partition(b, h, k);
```

```
    // We know  $b[h..j-1] \leq b[j] \leq b[j+1..k]$ 
```

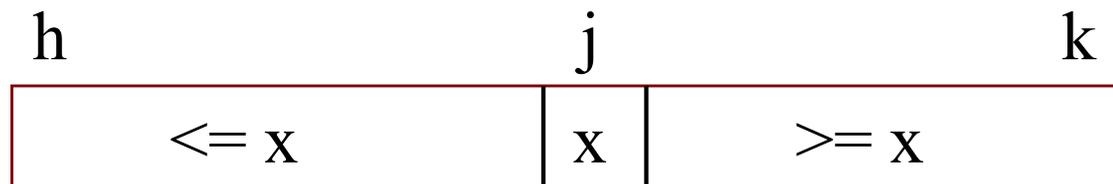
```
    // Sort  $b[h..j-1]$  and  $b[j+1..k]$ 
```

```
    QS(b, h, j-1);
```

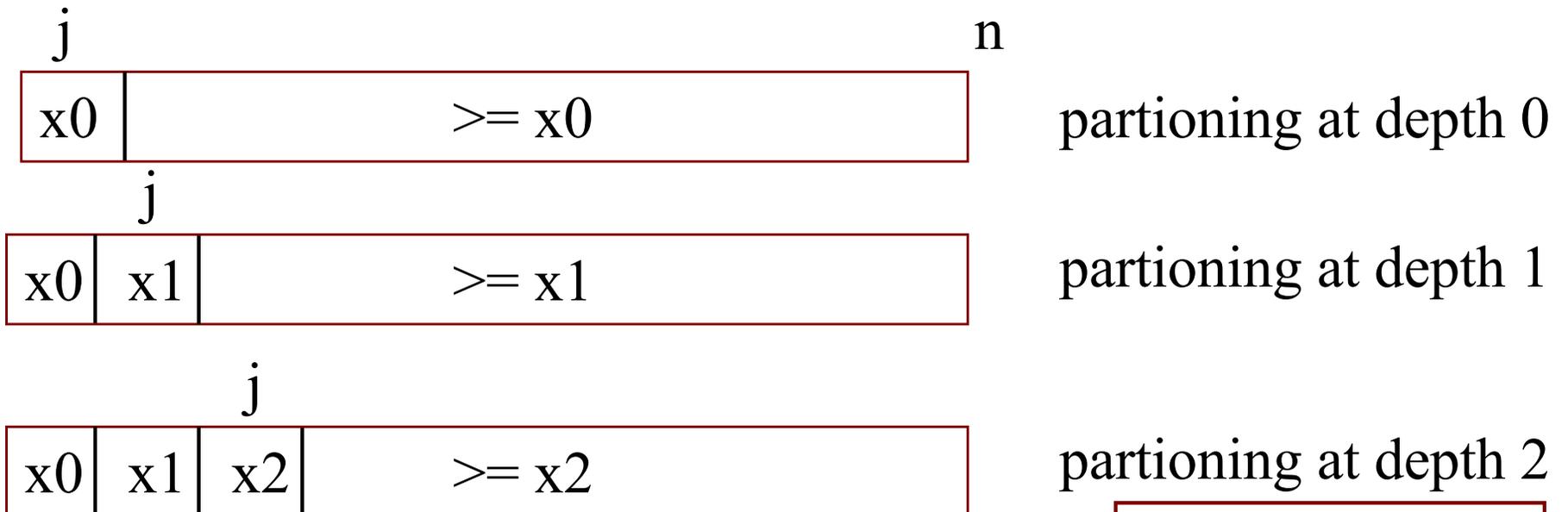
```
    QS(b, j+1, k);
```

```
}
```

Function does the partition algorithm and returns position j of pivot



Worst case quicksort: pivot always smallest value



```
/** Sort b[h..k]. */
```

```
public static void QS(int[] b, int h, int k) {  
    if (b[h..k] has < 2 elements) return;  
    int j= partition(b, h, k);  
    QS(b, h, j-1);    QS(b, j+1, k);  
}
```

Depth of
recursion: $O(n)$

Processing at
depth i : $O(n-i)$

$O(n*n)$

QuickSort complexity to sort array of length n

25

```
/** Sort b[h..k]. */  
public static void QS(int[] b, int h, int k) {  
    if (b[h..k] has < 2 elements) return;  
    int j= partition(b, h, k);  
    // We know b[h..j-1] <= b[j] <= b[j+1..k]  
    // Sort b[h..j-1] and b[j+1..k]  
    QS(b, h, j-1);  
    QS(b, j+1, k);  
}
```

Time complexity

Worst-case: $O(n^2)$

Average-case: $O(n \log n)$

Worst-case space: ?

What's depth of recursion?

Worst-case space: $O(n)$!

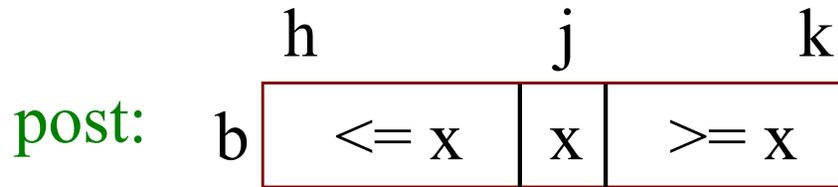
--depth of recursion can be n

Can rewrite it to have space $O(\log n)$

Show this at end of lecture if we have time

Partition. Key issue. How to choose pivot

26



Choosing pivot

Ideal pivot: the median,
since it splits array in half

But computing the median is
 $O(n)$, quite complicated

Popular heuristics: Use

- ◆ first array value (not so good)
- ◆ middle array value (not so good)
- ◆ Choose a random element (not so good)
- ◆ median of first, middle, last, values (often used)!

Performance

27

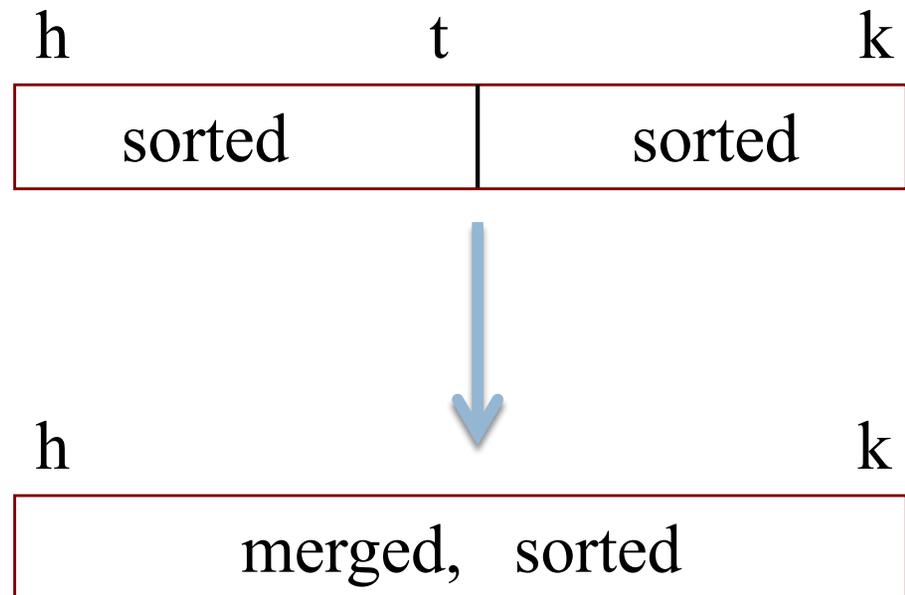
Algorithm	Ave time.	Worst-case time	Space	Stable?
Insertion sort	$O(n^2)$.	$O(n^2)$	$O(1)$	Yes
Selection sort	$O(n^2)$.	$O(n^2)$	$O(1)$	No
Quick sort	$O(n \log n)$.	$O(n^2)$	$O(\log n)^*$	No
Merge sort				

* The first algorithm we developed takes space $O(n)$ in the worst case, but it can be reduced to $O(\log n)$

Merge two adjacent sorted segments

28

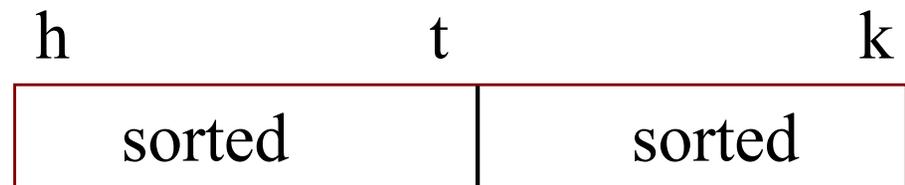
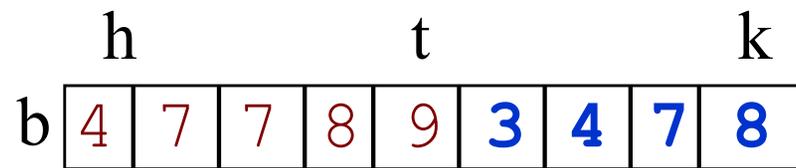
```
/* Sort b[h..k]. Precondition: b[h..t] and b[t+1..k] are sorted. */  
public static merge(int[] b, int h, int t, int k) {  
    Copy b[h..t] into a new array c;  
    Merge c and b[t+1..k] into b[h..k];  
}
```



Merge two adjacent sorted segments

29

```
/* Sort b[h..k]. Precondition: b[h..t] and b[t+1..k] are sorted. */  
public static merge(int[] b, int h, int t, int k) {  
}
```



Merge two adjacent sorted segments

30

// Merge sorted c and $b[t+1..k]$ into $b[h..k]$

pre: c

0	t-h
x	

 b

h	t	k
?		y

 x, y are sorted

post: b

h	k
x and y, sorted	

invariant: c

0	i	c.length
head of x	tail of x	

b

h	u	v	k
	?	tail of y	

↑
head of x and head of y, sorted

Merge

```
int i = 0;
int u = h;
int v = t+1;
while( i <= t-h){
    if(v <= k && b[v] < c[i]) {
        b[u] = b[v];
        u++; v++;
    }else {
        b[u] = c[i];
        u++; i++;
    }
}
```

pre: c

0	t-h
sorted	

 b

h	t	k
?	sorted	

post: b

h	k
sorted	

inv: 0

0	i	c.length
sorted	sorted	

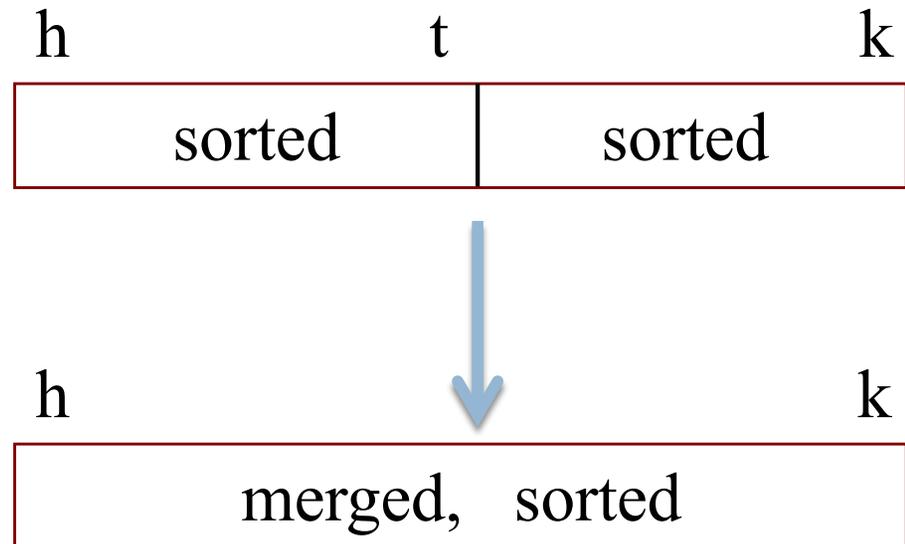
b

h	u	v	k
sorted	?	sorted	

Mergesort

32

```
/** Sort b[h..k] */  
public static void mergesort(int[] b, int h, int k) {  
    if (size b[h..k] < 2)  
        return;  
    int t = (h+k)/2;  
    mergesort(b, h, t);  
    mergesort(b, t+1, k);  
    merge(b, h, t, k);  
}
```



QuickSort versus MergeSort

33

```
/** Sort b[h..k] */  
public static void QS  
    (int[] b, int h, int k) {  
    if (k - h < 1) return;  
    int j= partition(b, h, k);  
    QS(b, h, j-1);  
    QS(b, j+1, k);  
}
```

```
/** Sort b[h..k] */  
public static void MS  
    (int[] b, int h, int k) {  
    if (k - h < 1) return;  
    MS(b, h, (h+k)/2);  
    MS(b, (h+k)/2 + 1, k);  
    merge(b, h, (h+k)/2, k);  
}
```

One processes the array then recurses.
One recurses then processes the array.

Performance

34

Algorithm	Ave time.	Worst-case time	Space	Stable?
Insertion sort	$O(n^2)$.	$O(n^2)$	$O(1)$	Yes
Selection sort	$O(n^2)$.	$O(n^2)$	$O(1)$	No
Quick sort	$O(n \log n)$.	$O(n^2)$	$O(\log n)^*$	No
Merge sort	$O(n \log n)$.	$O(n \log n)$	$O(n)$	Yes

* The first algorithm we developed takes space $O(n)$ in the worst case, but it can be reduced to $O(\log n)$

Sorting in Java

35

- `Java.util.Arrays` has a method `sort(array)`
 - ▣ implemented as a collection of overloaded methods
 - ▣ for primitives, `sort` is implemented with a version of quicksort
 - ▣ for Objects that implement `Comparable`, `sort` is implemented with `timSort`, a modified mergesort developed in 1993 by Tim Peters
- Tradeoff between speed/space and stability/performance guarantees