http://www.cs.cornell.edu/courses/cs1110/2019sp

Lecture 26: Sorting

CS 1110

Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

Plan of Attack

- **Insertion Sort**
- Partition
- Quick Sort

Searching is a good motivation for Sorting

Example: 500 CS 1110 Prelims have been scanned

Grading Session: "Hey, this scan is hard to read."

Task: go through 500 Exams, find the bad scan

Do you want this job?

Are the exams in any order? No….

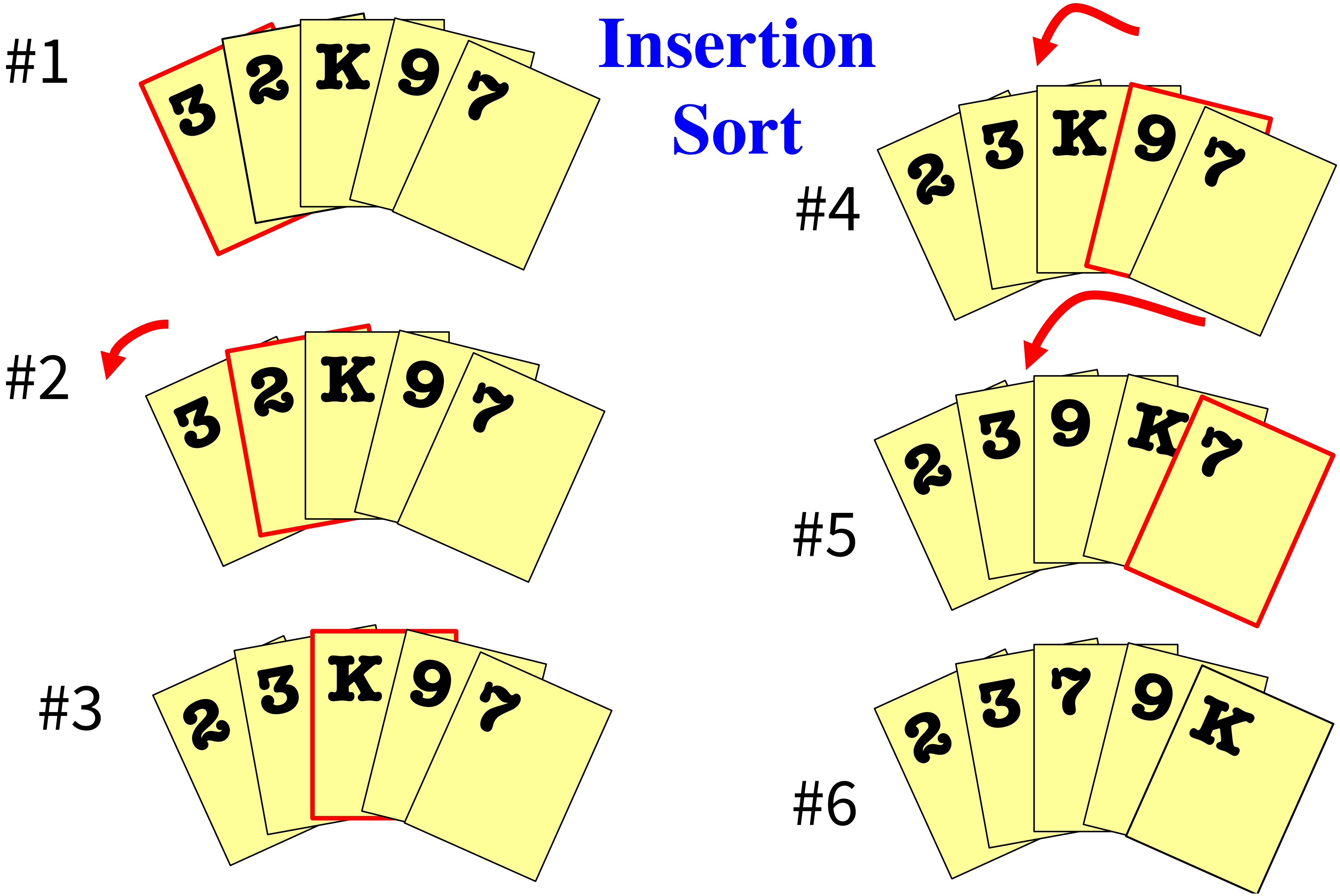
Fine, go through them all.

10 minutes later "Hey, this scan is hard to read…"

Now you *really* wish they were in order…

Sorting: Arranging in Ascending Order

Insertion Sort



Insertion Sort

0 n

PRE: b | *? (unknown values)* |

k = 0

0 k n

INV: b | *sorted* | *? (unknown)* |

```
while k < n:
    # Push b[k] down into its
    # sorted position in b[0..k]
    k = k+1
```

0						k
2	4	4	6	6	7	5

0						k
2	4	4	5	6	6	7

0 n

POST: b | *sorted* |

Insertion Sort: Moving into Position

```
def push_down(b, k):
    while k > 0:
        if b[k-1] > b[k]:
            swap(b,k-1,k)
        k = k-1

k = 0
while k < n:
    push_down(b,k)
    k = k+1
```

k=6

0						k	
2	4	4	6	6	7	5	

How many swaps will there be?

A: 1
B: 2
C: 3
D: 4
E: 5

Insertion Sort: Moving into Position

```
def push_down(b, k):
    while k > 0:
        if b[k-1] > b[k]:
            swap(b,k-1,k)
        k = k-1

k = 0
while k < n:
    push_down(b,k)
    k = k+1
```

k=6

0						k	
2	4	4	6	6	**7**	**5**	

k=5

0					k		
2	4	4	6	6	5	7	

k=4

0				k			
2	4	4	6	5	6	7	

k=3

0			k				
2	4	4	5	6	6	7	

3 swaps!

The Importance of Helper Functions

```
def push_down(b, k):
    while k > 0:
        if b[k-1] > b[k]:
            swap(b,k-1,k)
        k = k-1

k = 0
while k < n:
    push_down(b,k)
    k = k+1
```

VS

Can you understand all this code below?

```
k = 0
while k < n:
    j = k
    while j > 0:
        if b[j-1] > b[j]:
            temp = b[j]
            b[j] = b[j-1]
            b[j-1] = temp
        j = j -1
    k = k +1
```

Also: Is this how you want to sort 500 exams?

Algorithm Complexity

```
def push_down(b, k):
    while k > 0:
        if b[k-1] > b[k]:
            swap(b,k-1,k)
        k = k-1

k = 0
while k < n:
    push_down(b,k)
    k = k+1
```

Nested loops multiply the number of operations required. We need to compare b[k] to all elements. ~*n* operations

Iterating through a sequence of length *n* requires *n* operations: push_down called *n* times

Approximately how many operations/swaps does this take?

A: ~ 1 B: $\sim n$
C: $\sim n^2$ D: $\sim n^3$
E: I don't know

Clicker Answer: Algorithm Complexity

```
def push_down(b, k):
    while k > 0:
        if b[k-1] > b[k]:
            swap(b,k-1,k)
        k = k-1

k = 0
while k < n:
    push_down(b,k)
    k = k+1
```

Approximately how many operations/swaps does this take?

A: ~ 1 operation
B: ~ n operations
C: ~ n^2 operations **CORRECT**
D: ~ n^3 operations
E: I don't know

Actual Algorithm Complexity

```
def push_down(b, k):
    while k > 0:
        if b[k-1] > b[k]:
            swap(b,k-1,k)
        k = k-1

k = 0
while k < n:
    push_down(b,k)
    k = k+1
```

Each call to push down must go through a longer and longer series of swaps

Total Swaps:

$0 + 1 + 2 + \ldots (n-1)$

$= n * (n-1)/2$

Insertion sort requires n*n operations

https://www.youtube.com/watch?v=xxcpvCGrCBc

Plan of Attack

- Insertion Sort
- **Partition**
  - Overview in class
  - Details (optional!) are at the end of this lecture
- Quick Sort

Partition

What if we had an algorithm that could partition a list segment based on some value?

called the pivot value

```
def partition(b, h, k):
```

change:

	h								k	
b	3	5	4	1	6	2	3	8	1	

into

	h			i					k	
b	1	2	1	3	5	4	6	3	8	

Like separating positives from negatives but instead separating by the first value in the segment.

We can use this to make a faster sort!

Plan of Attack

- Insertion Sort
- Partition
- **Quick Sort**

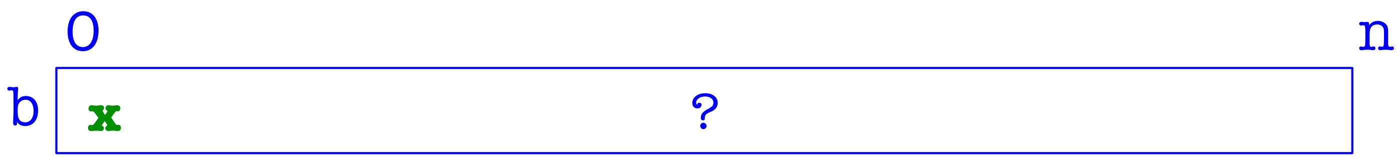
Let's partition the exams into 2 piles: A-M & N-Z.

Now let's partition the A-M pile into A-F & G-M

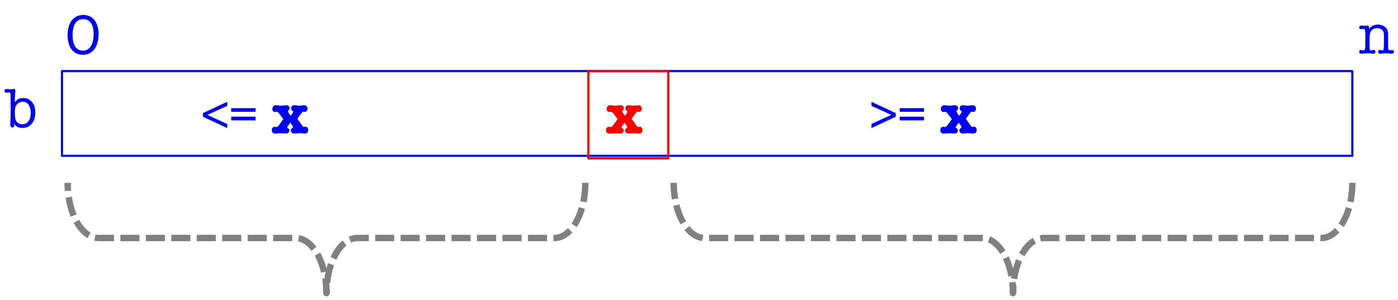
Now let's partition the A-F pile into A-C & D-F

eventually the exams will all be sorted!

Sorting with Partitions



- **Idea:** Pick a *pivot* element **x**
- Partition sequence into <= x and >= x



Now Partition this

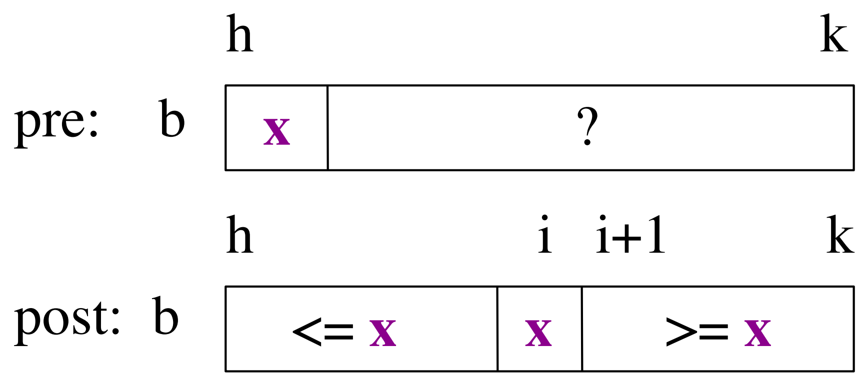
and this, too

Keep recursing…

https://www.youtube.com/watch?v=m1PS8IR6Td0

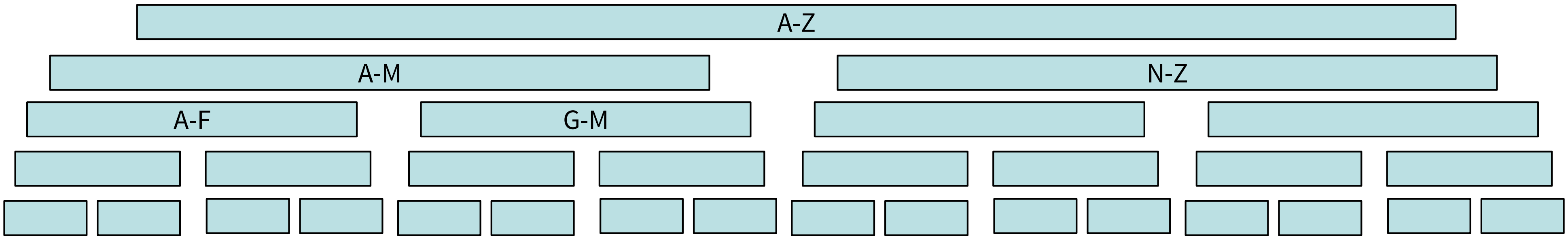
QuickSort

```
def quick_sort(b, h, k):
    """Sort the array fragment b[h..k]"""
    if  k<=h:
        return
    i = partition(b, h, k)
    # INV: b[h..i-1] <= b[i] <= b[i+1..k]
    # Sort b[h..i-1]  and  b[i+1..k]
    quick_sort (b, h, i-1)
    quick_sort (b, i+1, k)
```

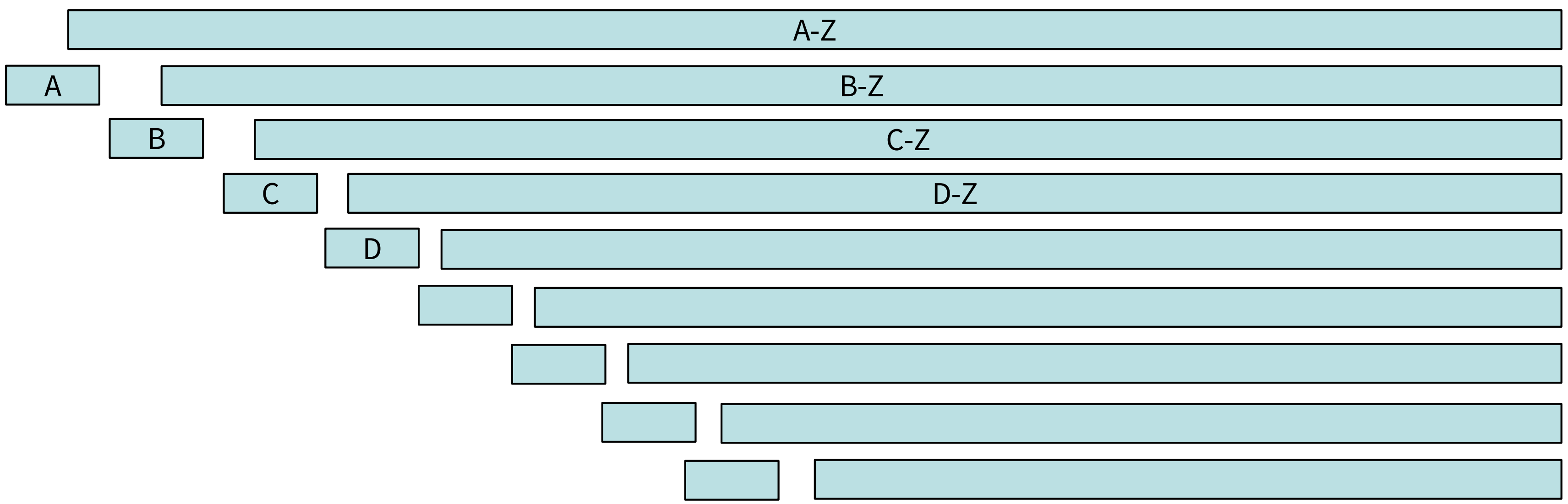


How Fast is QuickSort?

If you're lucky, each partition will split the list in half. **Runtime: n * log(n)**



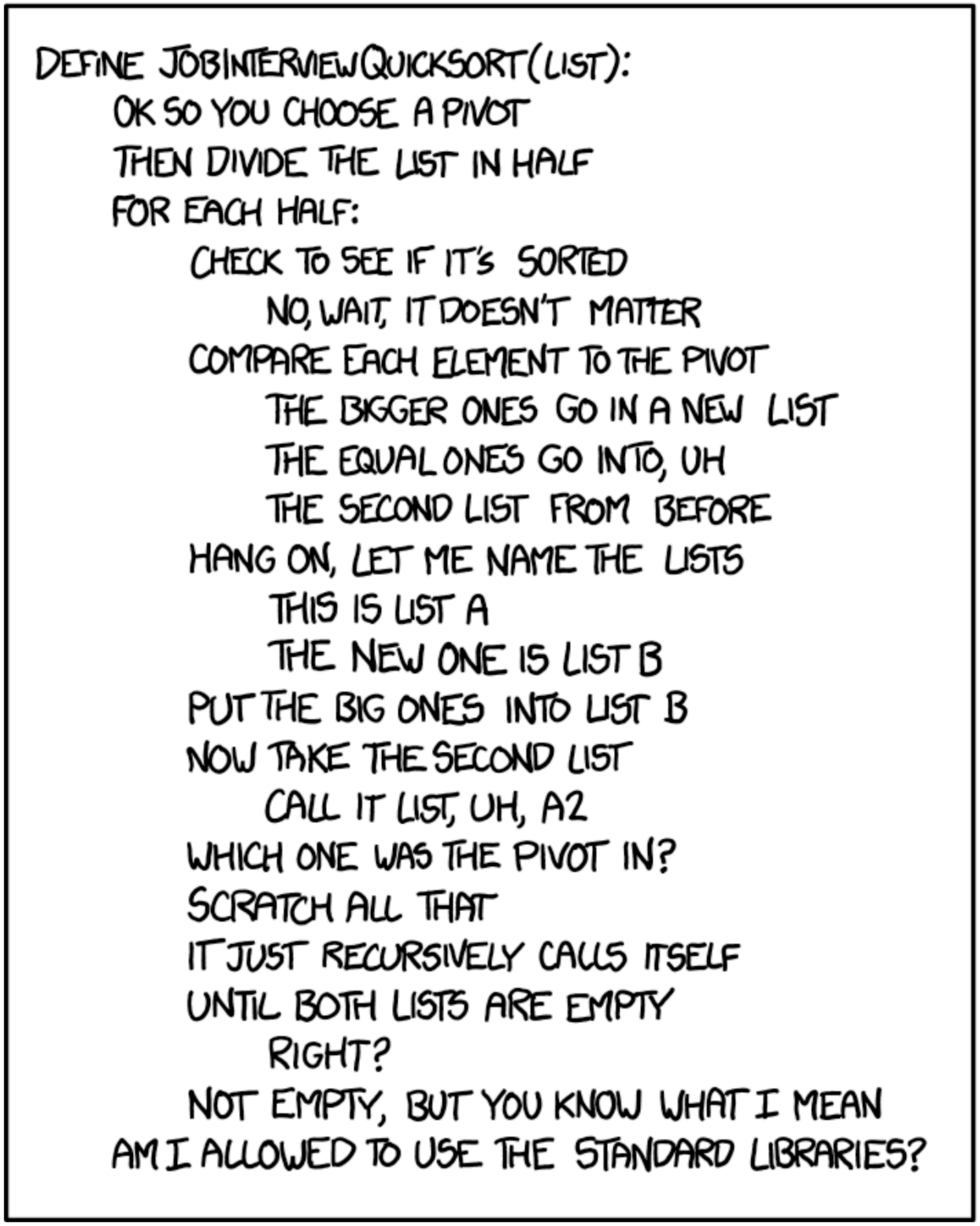
If you're **not** lucky, each partition removes only 1 element from the list. **Runtime = n * n**



In practice, you get lucky.

and on and on and on…

Quicksort in the real world



https://xkcd.com/1185/

Appendix: Partition Details

You are **not** responsible for knowing the details of the following slides but they are a good (but difficult*) case study of how to develop an algorithm using loop invariants

* certainly more difficult than anything we would ask you on the Final Exam

OPTIONAL

Partition Algorithm

- Given a list segment b[h..k] with some pivot value x in b[h]:

pre: b

	h		k	
	x	?		

- Swap elements of b[h..k] and store in **i** to satisfy postcondition:

post: b

	h	i	i+1	k	
	<= x	x	>= x		

Example:

change: b

	h							k	
	3	5	4	1	6	2	3	8	1

into b

	h			i					k
	1	2	1	**3**	5	4	6	3	8

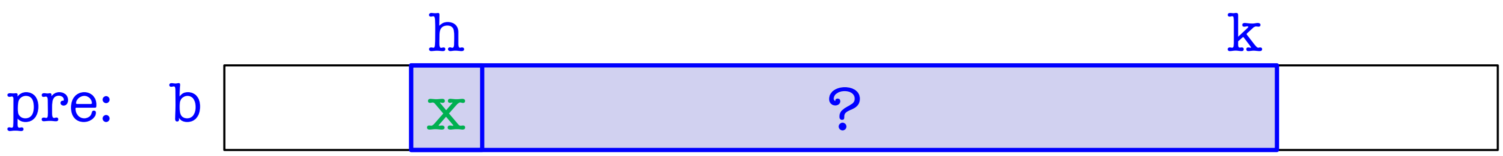
x

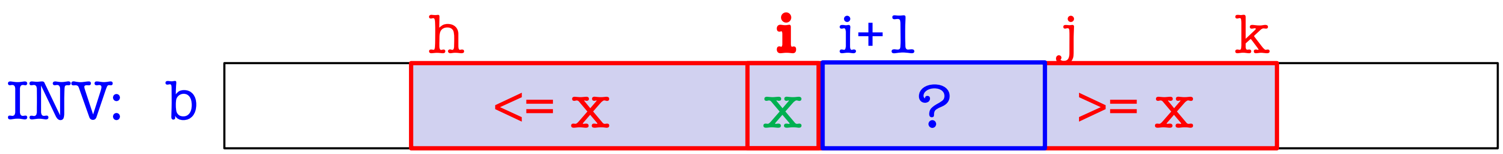
- Called the pivot value
- not a variable
- = whatever value is in b[h]

OPTIONAL

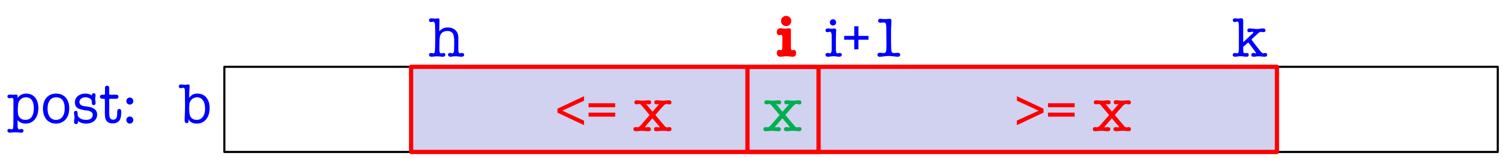
Partition: What's the Invariant?

- Given a list segment `b[h..k]` with some pivot value `x` in `b[h]`:





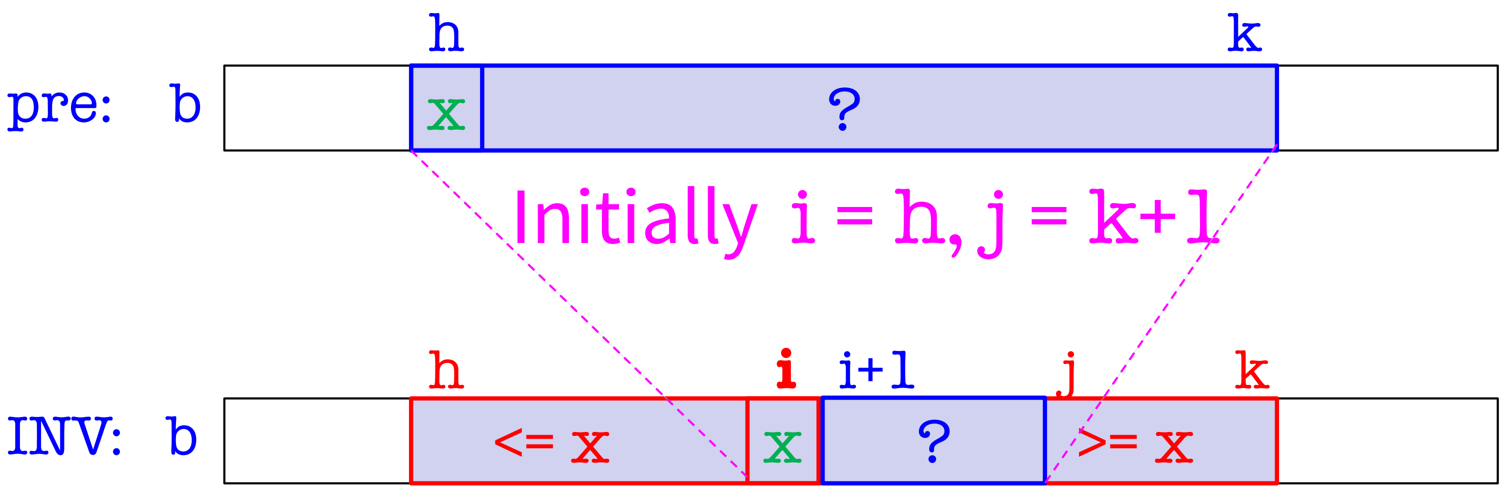
- Swap elements of `b[h..k]` and store in **i** to satisfy postcondition:



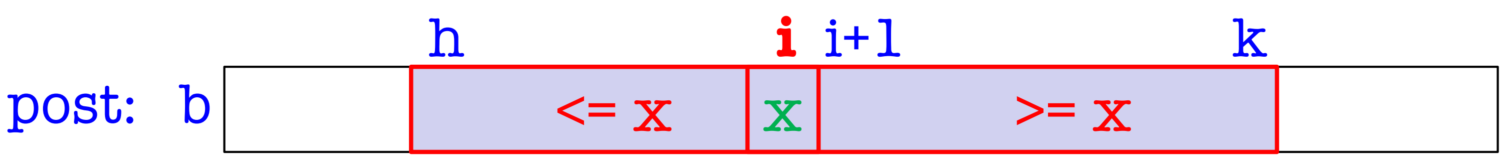
OPTIONAL

Partition: What's the Invariant?

- Given a list segment b[h..k] with some pivot value x in b[h]:



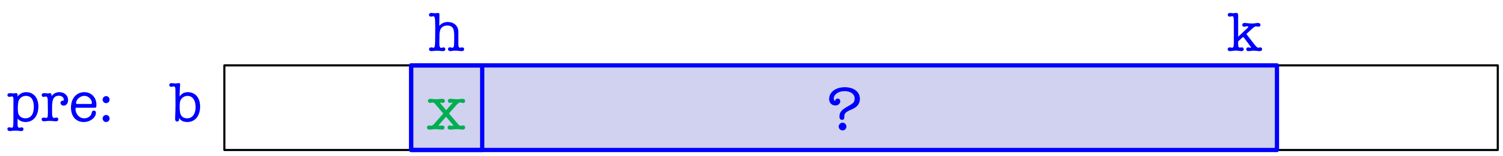
- Swap elements of b[h..k] and store in **i** to satisfy postcondition:

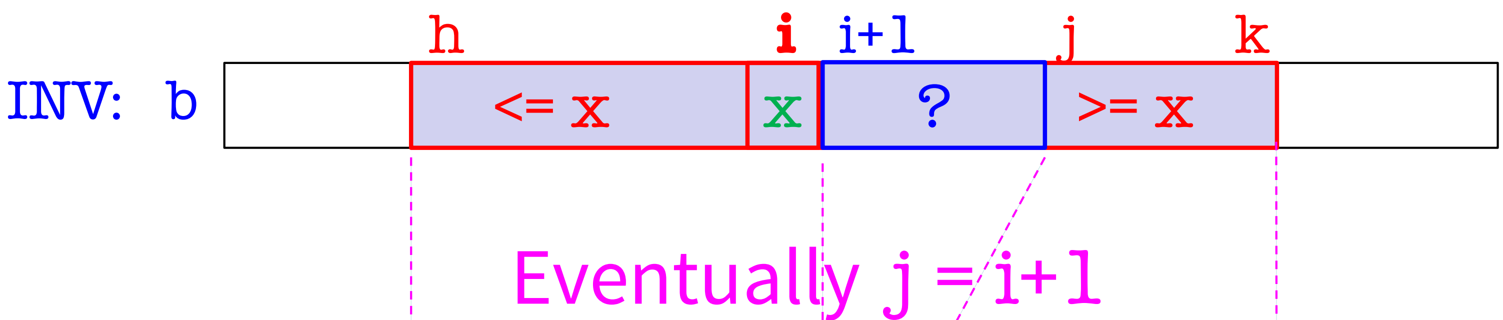


OPTIONAL

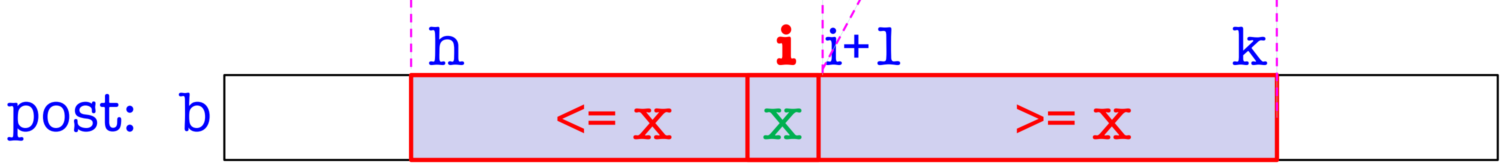
Partition: What's the Invariant?

- Given a list segment `b[h..k]` with some pivot value `x` in `b[h]`:





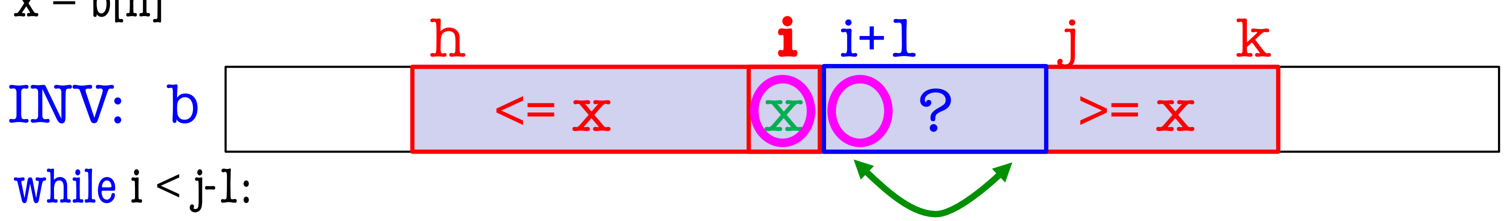
- Swap elements of `b[h..k]` and store in **`i`** to satisfy postcondition:



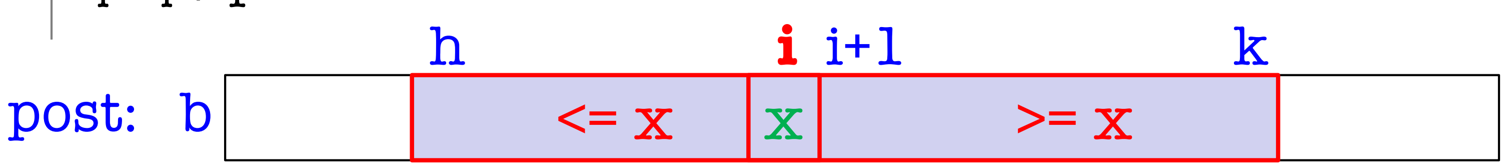
OPTIONAL

Partition Algorithm Implementation


```
def partition(b, h, k):
    i = h
    j = k+1
    x = b[h]
```



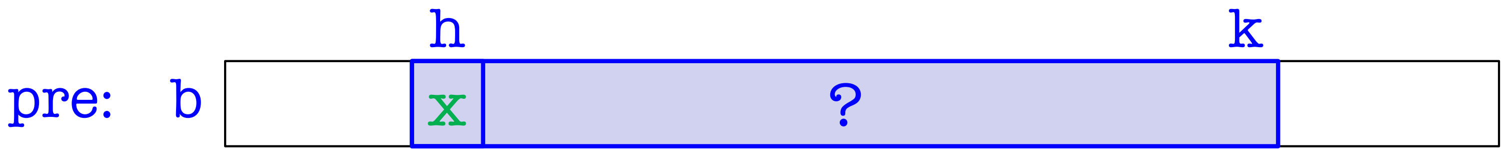

```
    while i < j-1:
        if b[i+1] >= x:
            # Move b[i+1] to end of block.
            swap(b,i+1,j-1)
            j = j - 1
        else:   # b[i+1] < x
            swap(b,i,i+1)
            i = i + 1
```



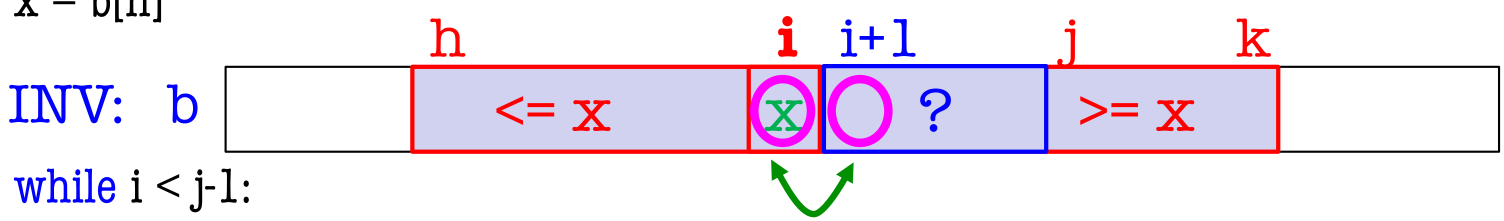

```
    return i
```

OPTIONAL

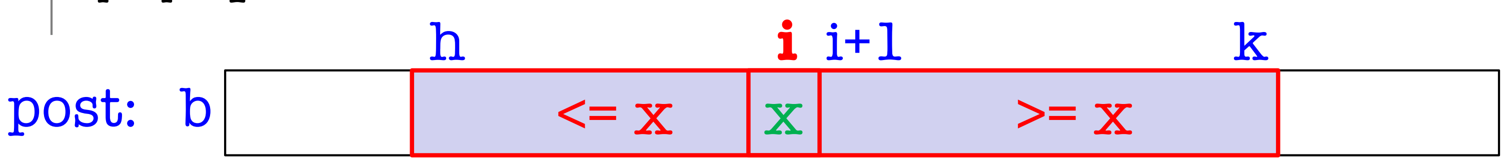
Partition Algorithm Implementation




```
def partition(b, h, k):
    i = h
    j = k+1
    x = b[h]
```




```
    while i < j-1:
        if b[i+1] >= x:
            # Move to end of block.
            swap(b,i+1,j-1)
            j = j - 1
        else:   # b[i+1] < x
            swap(b,i,i+1)
            i = i + 1
```




```
    return i
```