http://www.cs.cornell.edu/courses/cs1110/2018sp

Lecture 26: Sorting

CS 1110

Introduction to Computing Using Python

Cornell CIS
COMPUTING AND INFORMATION SCIENCE

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

Announcements

- Academic Integrity:
  - Remember to cite your sources
- Assignment 5
  - Due 11:59pm on ***<u>**Wednesday**</u>*** May 9th
- Last Lab due next week by 10 mins into your Lab Section.
- Final Exam
  - May 17th, 9am-11:30am
  - **Location**: Barton Hall Central and East
  - Conflict assignment on CMS

Announcements

- Assignment 5 and Print Statements
- Where did your debugging print statements go?
  - Line 398 in `a5_unochecks.py` says:

```
# Comment out this line to allow print statements
```

So please go comment this out if you want your print statements to show

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.

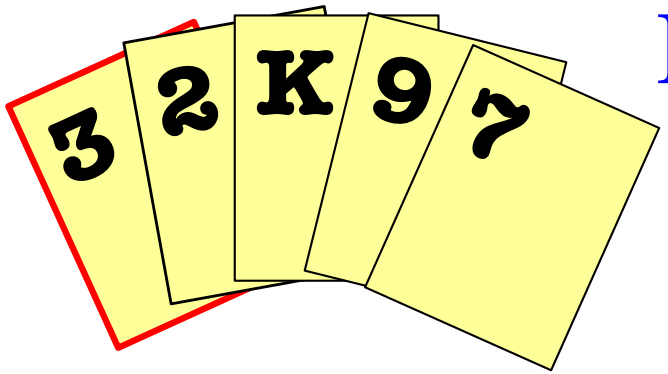
10 minutes later "Hey, this scan is hard to read…"

Now you *really* wish they were in order…

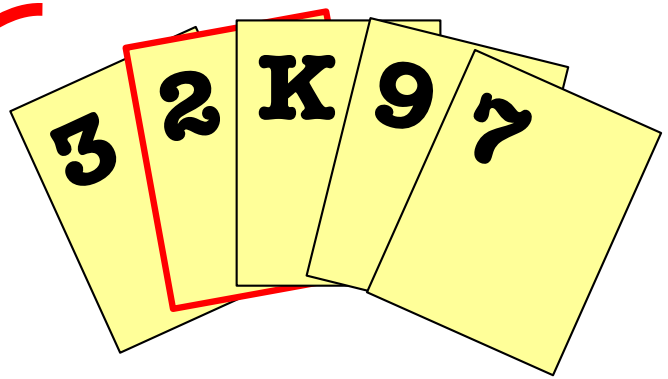
Sorting: Arranging in Ascending Order

Insertion Sort

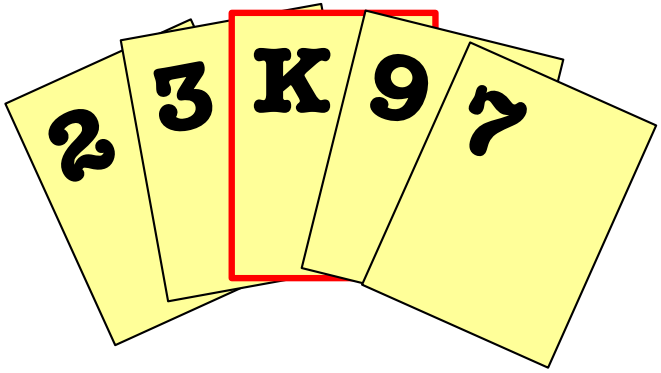
#1



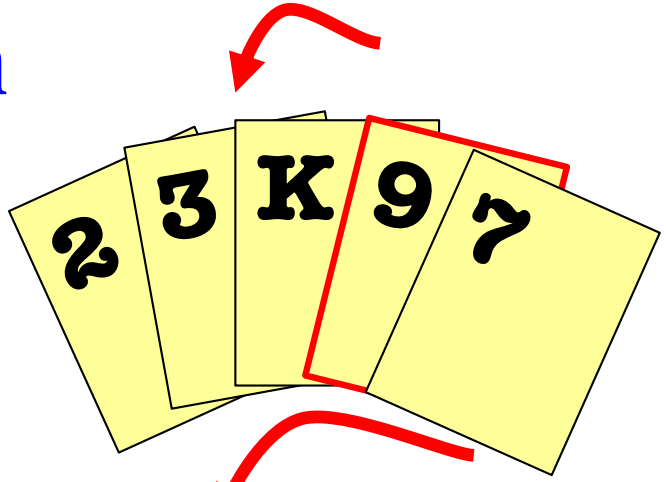
#2



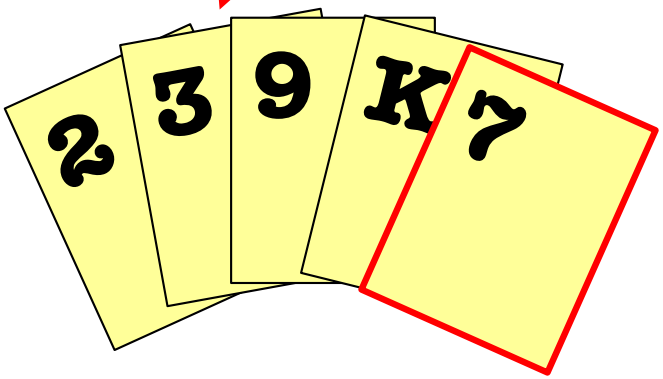
#3



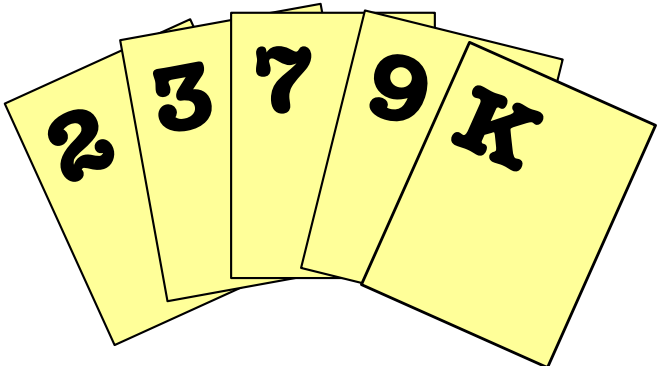
#4



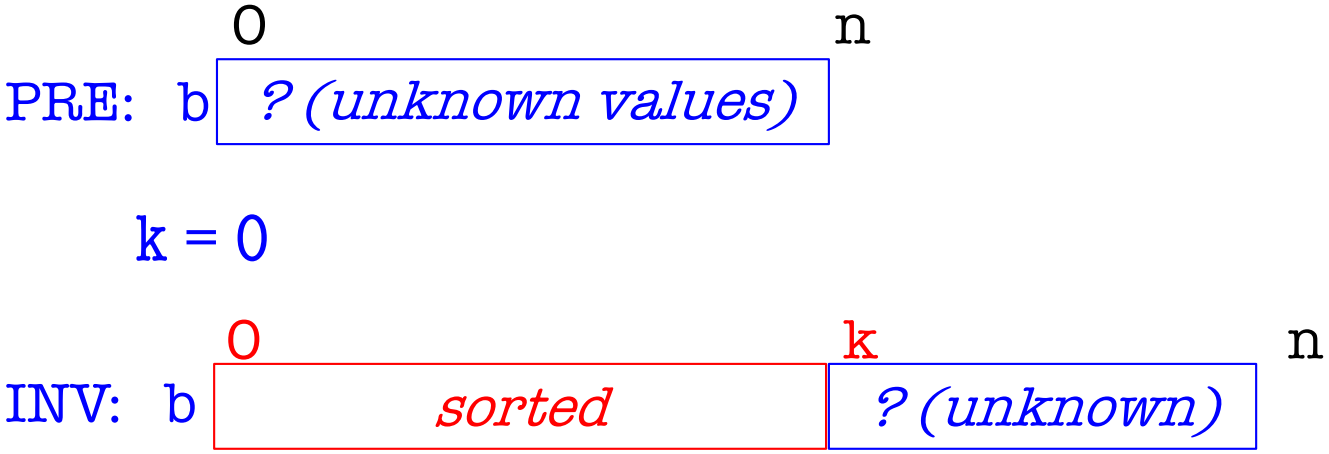
#5



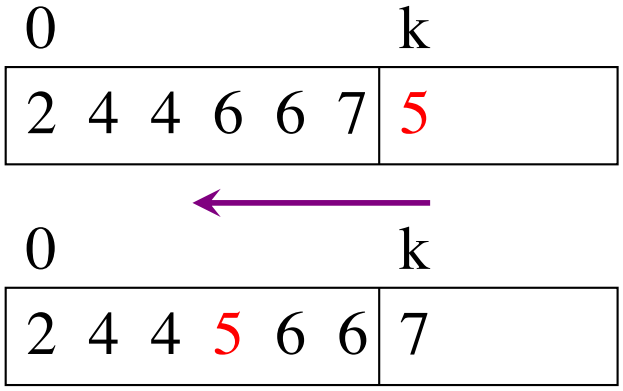
#6



Insertion Sort




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

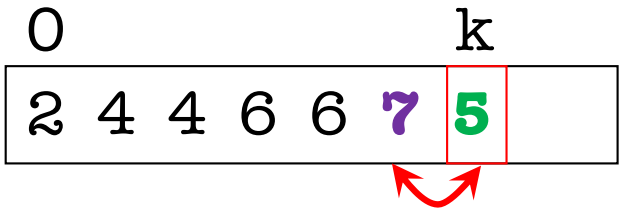


Insertion Sort: Moving into Position

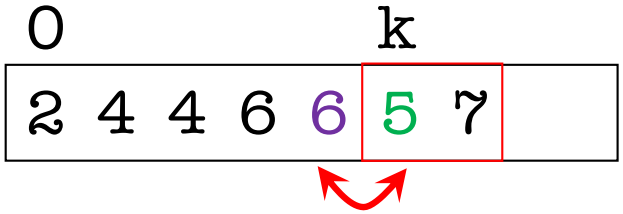
```
def push_down(b, k):
    while k > 0:
        if b[k-1] > b[k]:
            swap(b,k-1,k)
        k = k-1


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

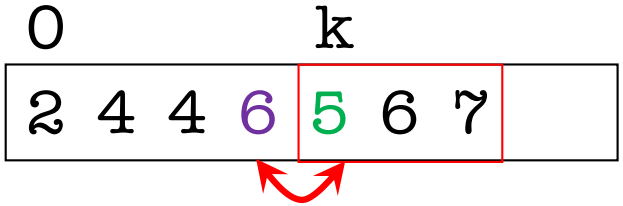
k=6



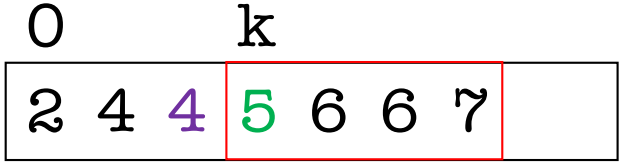
k=5



k=4



k=3



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

```
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
```

Can you understand all this code below?

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

Q: 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 does this take?

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

A: 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 does this take?

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

Total Swaps: 0 + 1 + 2 + 3 + … (n-1)
= (n-1)*n/2

Insertion sort requires n*n operations

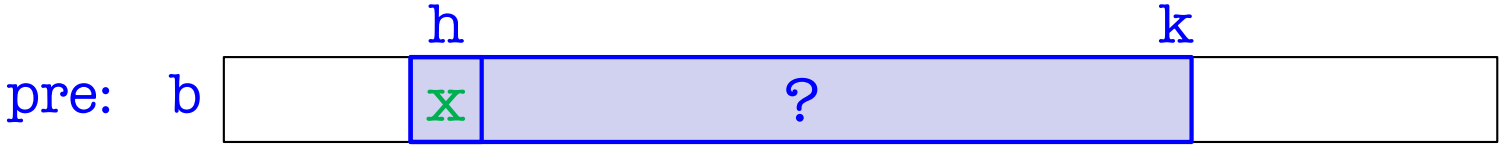
https://www.youtube.com/watch?v=xxcpvCGrCBc

Plan of Attack

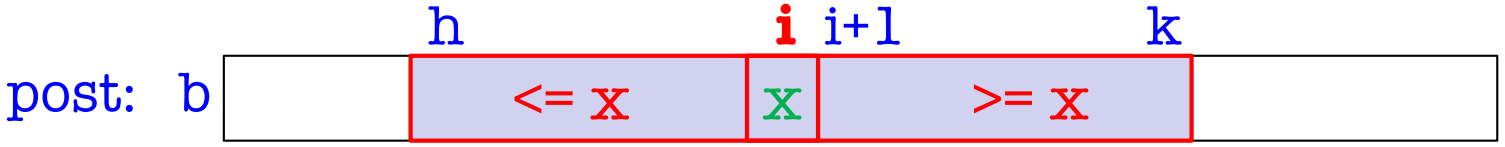
- Insertion Sort
- **Partition**
- Quick Sort

Partition Algorithm

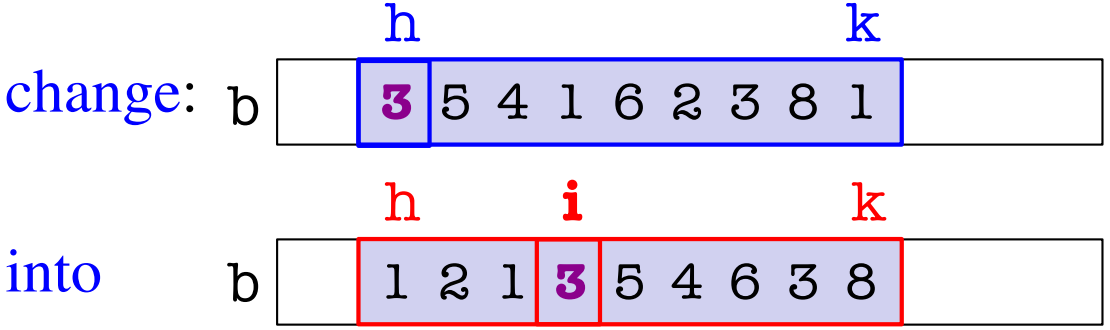
- Given a list segment b[h..k] with some pivot value x in b[h]:



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



Example:

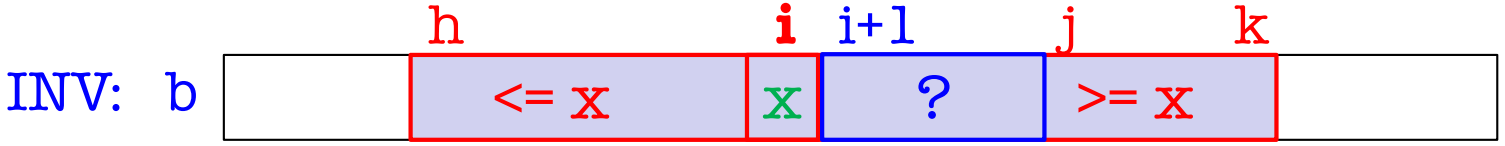


x
- Called the pivot value
- not a variable

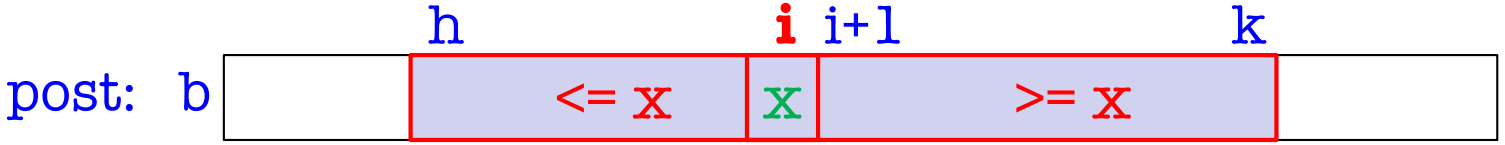
= whatever value is in b[h]

Partition: What's the Invariant?

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

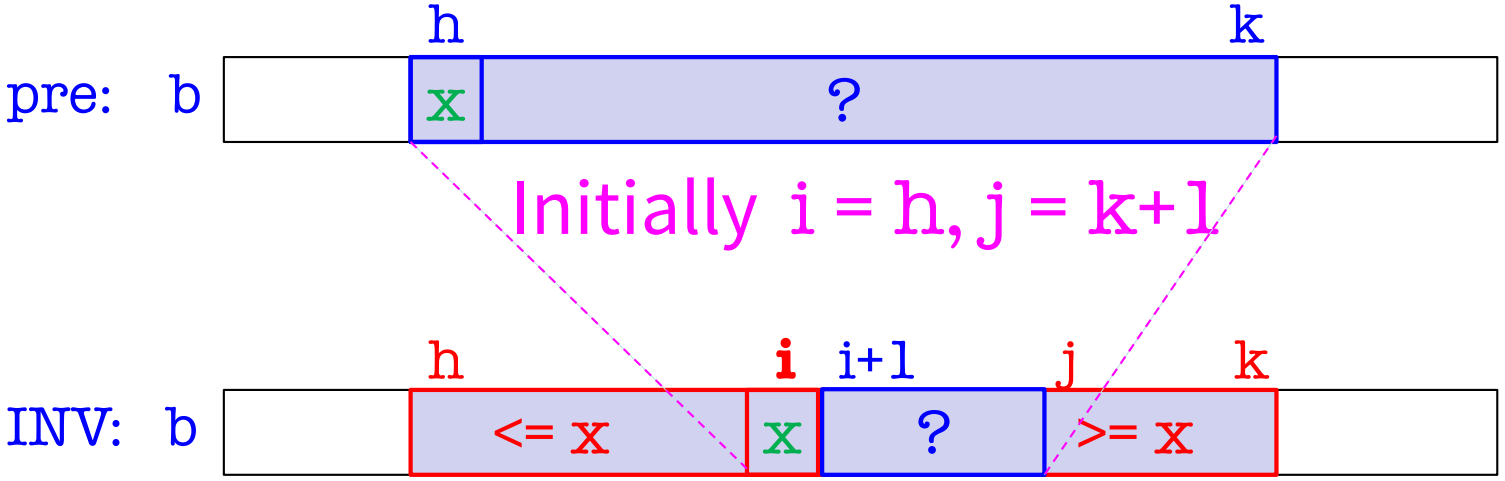


- Swap elements of b[h..k] and store in i to satisfy postcondition:

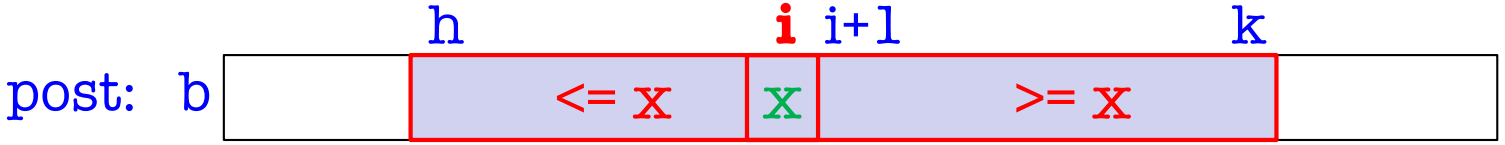


Partition: What's the Invariant?

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

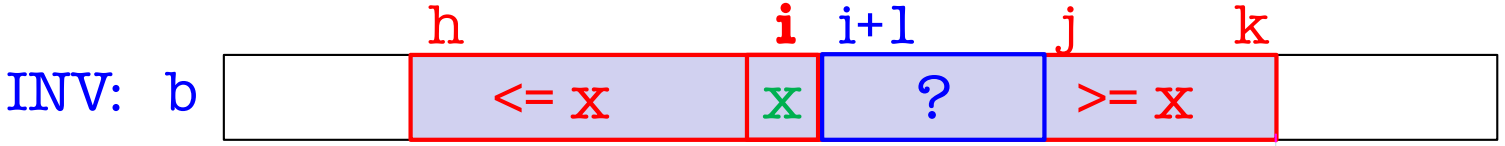


- Swap elements of b[h..k] and store in i to satisfy postcondition:



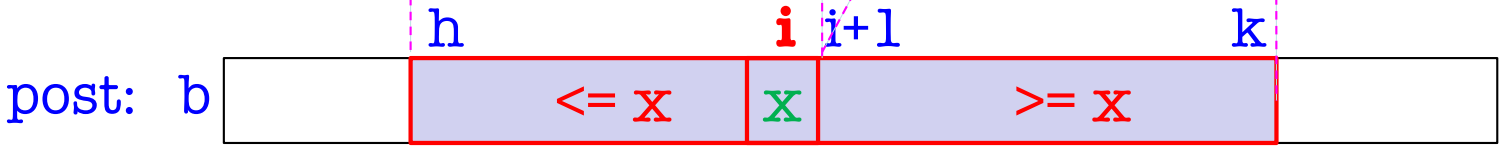
Partition: What's the Invariant?

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

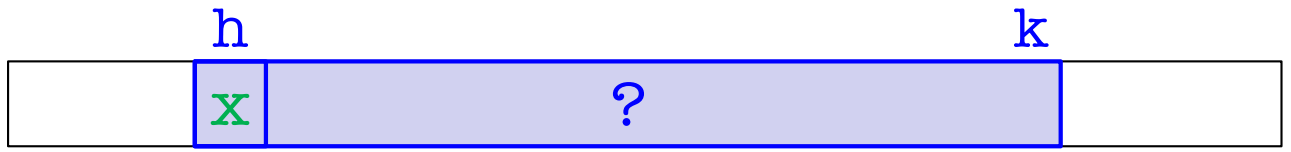


Eventually j = i+1

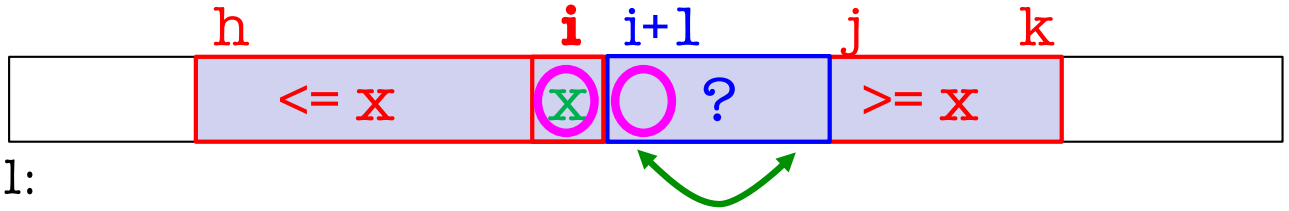
- Swap elements of b[h..k] and store in i to satisfy postcondition:



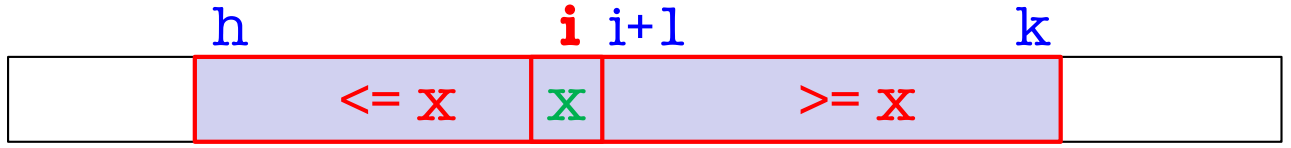
Partition Algorithm Implementation




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



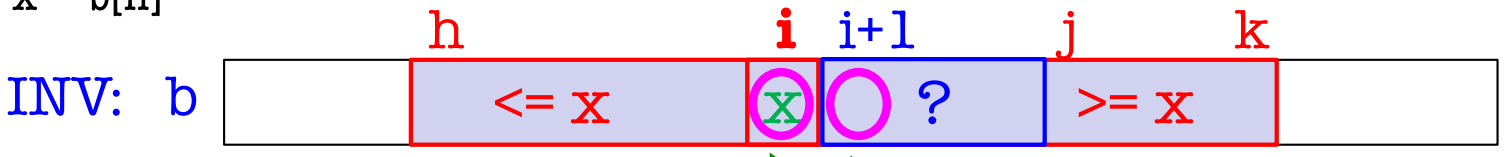

```
    while i < j-1:
        if b[i+1] >= x:
            # Move b[i+1] to end of block.
            swap(b,i+1,j-1)
            j = j - 1
        else:   # b[i+1] < x
            swap(b,i,i+1)
            i = i + 1
```



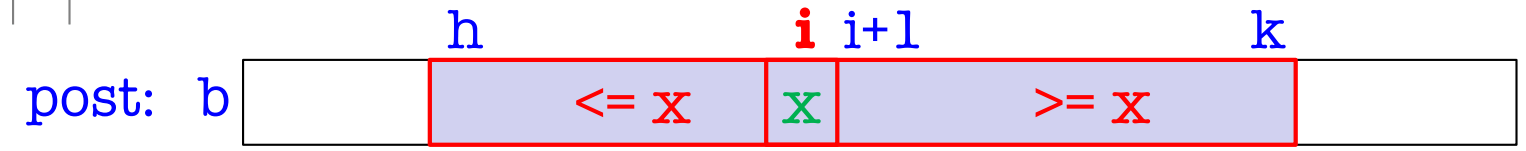

```
    return i
```

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
```

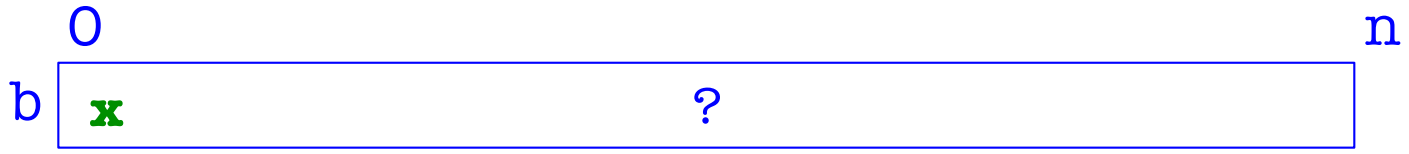



```
    return i
```

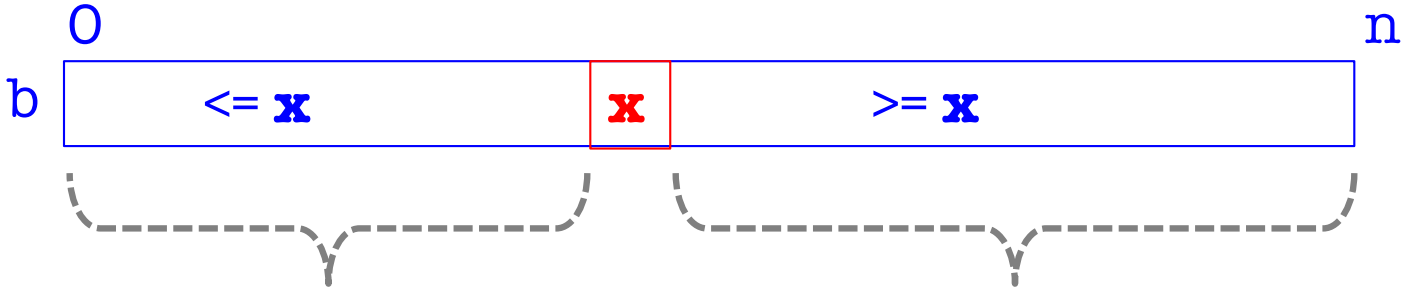
Plan of Attack

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

Sorting with Partitions



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



Keep recursing…

https://www.youtube.com/watch?v=m1PS8IR6TdO

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)
```