CS473 - Algorithms I

Lecture 5
Quicksort

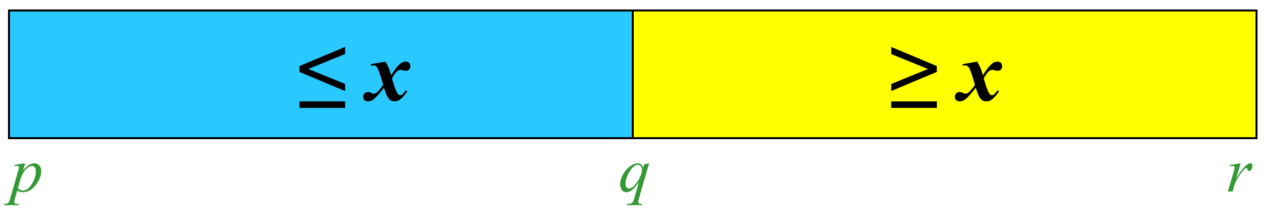
Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Quicksort

- One of the most-used algorithms in practice
- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm
- In-place algorithm
  - The additional space needed is O(1)
  - The sorted array is returned in the input array
  - *Reminder: Insertion-sort is also an in-place algorithm, but Merge-Sort is not in-place.*
- Very practical

Quicksort

1. Divide: Partition the array into 2 subarrays such that elements in the lower part ≤ elements in the higher part



2. Conquer: Recursively sort 2 subarrays
3. Combine: Trivial (because in-place)

- Key: Linear-time ($\Theta(n)$) partitioning algorithm

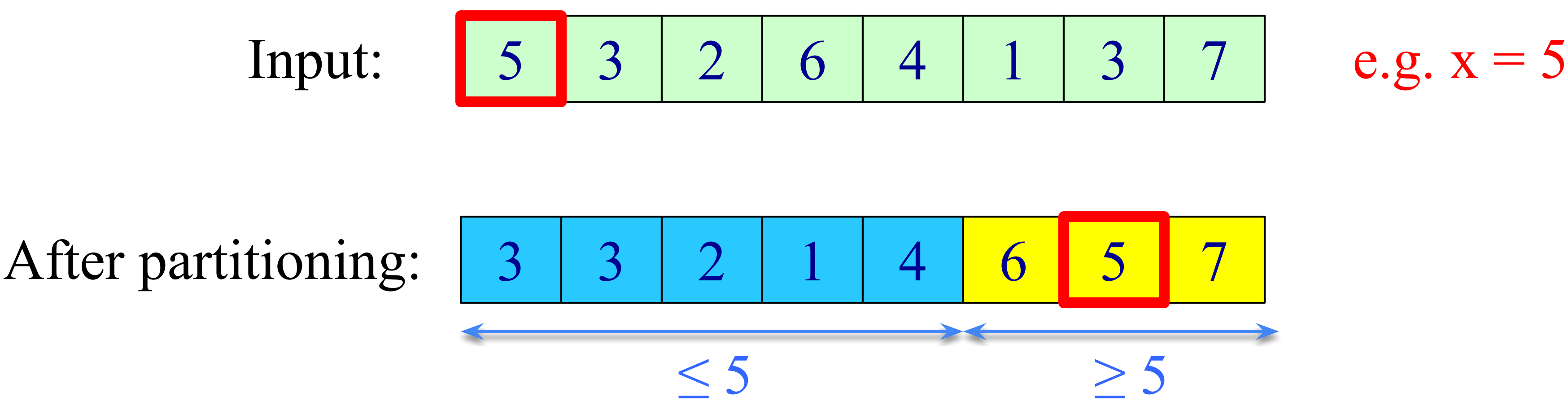
Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Divide: Partition the array around a pivot element

1. Choose a **pivot** element x
2. Rearrange the array such that:

   Left subarray: All elements ≤ x

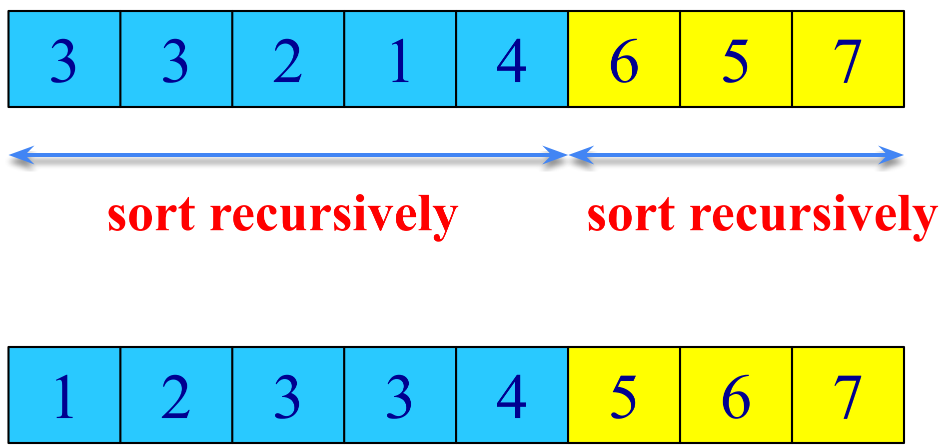
   Right subarray: All elements ≥ x



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Conquer: Recursively Sort the Subarrays

Note: Everything in the left subarray ≤ everything in the right subarray



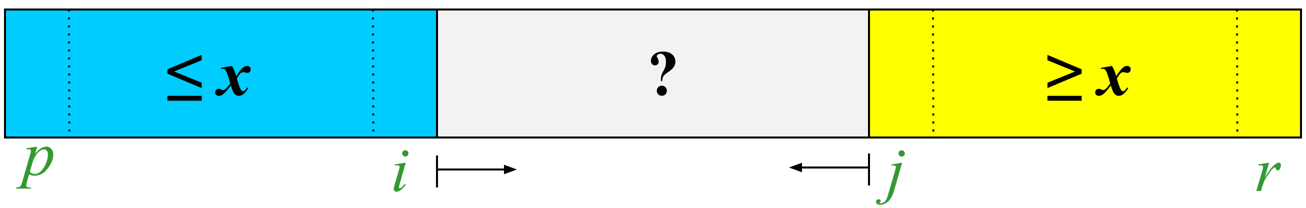
After conquer:

Note: Combine is trivial after conquer. Array already sorted.

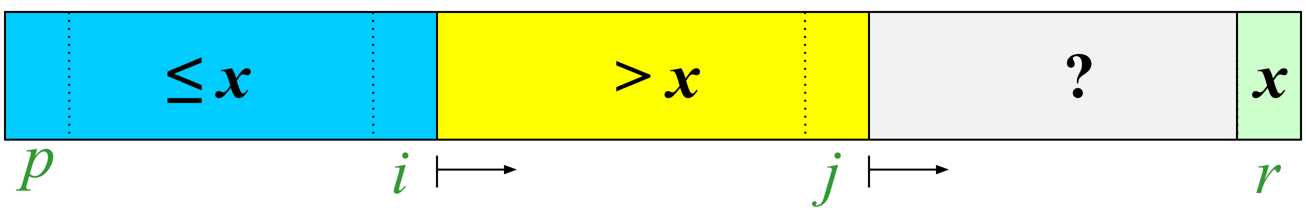
Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Two partitioning algorithms

1. **Hoare's algorithm:** Partitions around the first element of subarray (*pivot* = x = A[p])



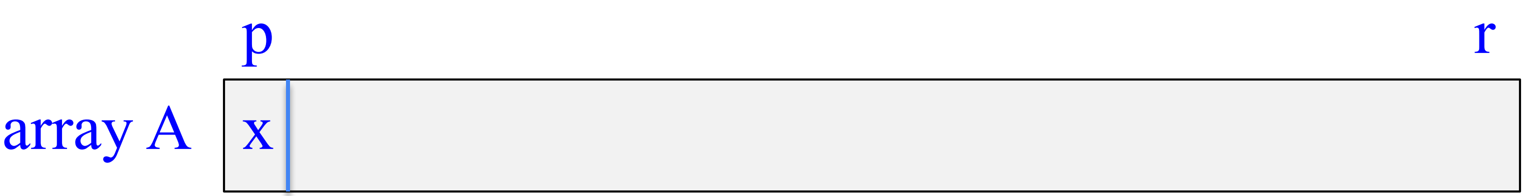
2. **Lomuto's algorithm:** Partitions around the last element of subarray (*pivot* = x = A[r])



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

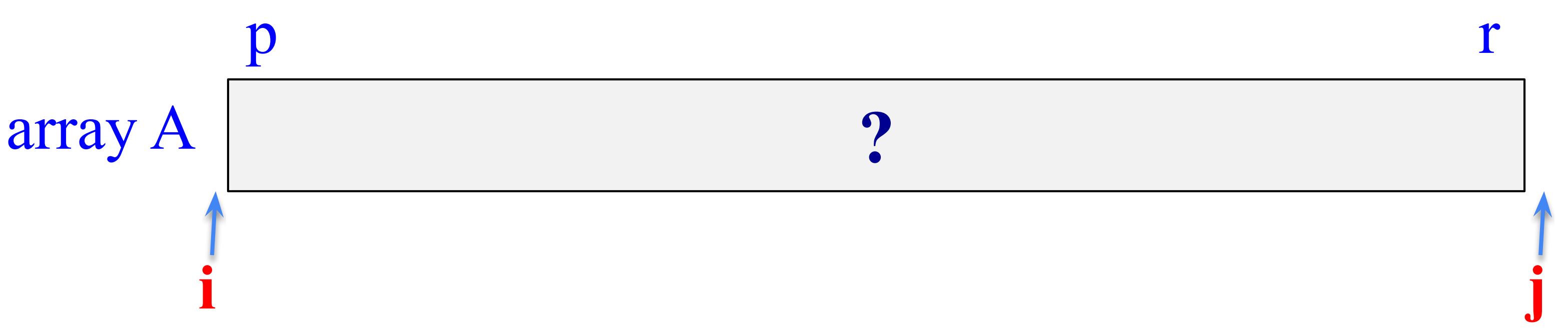
Hoare's Partitioning Algorithm

1. Choose a pivot element: pivot = x = A[p]
2. Grow two regions:
   from left to right: A[p..i]
   from right to left: A[j..r]
   such that:
   every element in A[p…i] $\leq$ pivot
   every element in A[j…r] $\geq$ pivot



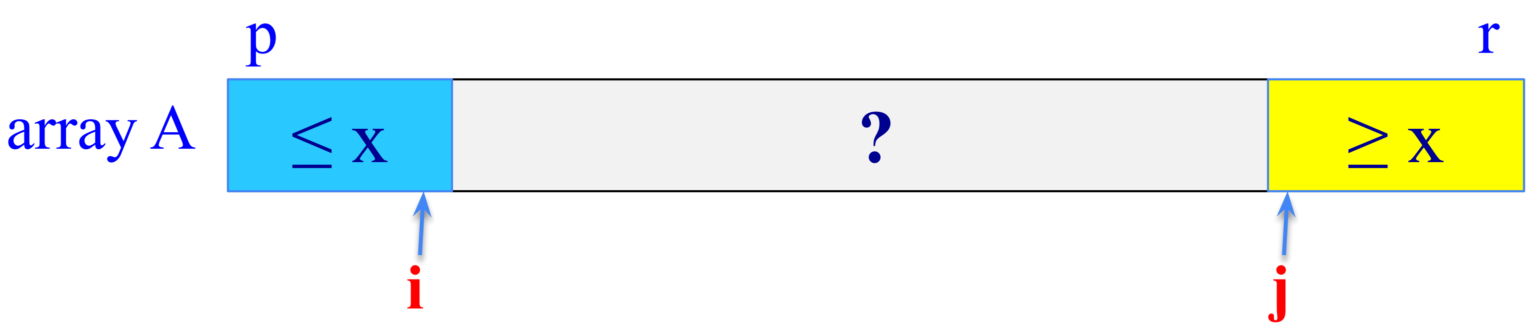
Hoare's Partitioning Algorithm

1. Choose a pivot element: pivot = x = A[p]
2. Grow two regions:
   from left to right: A[p..i]
   from right to left: A[j..r]
   such that:
   every element in A[p…i] $\leq$ pivot
   every element in A[j…r] $\geq$ pivot



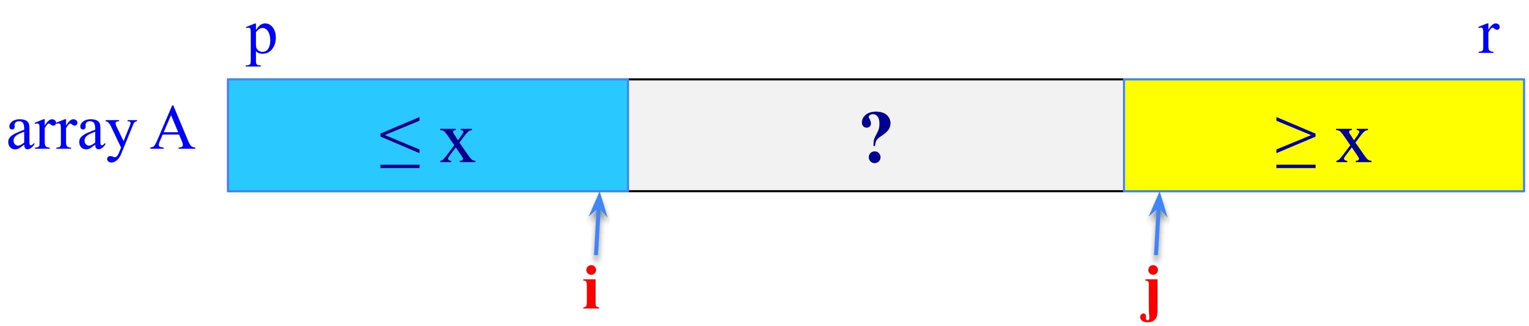
Hoare's Partitioning Algorithm

1. Choose a pivot element: pivot = x = A[p]
2. Grow two regions:
   from left to right: A[p..i]
   from right to left: A[j..r]
   such that:
   every element in A[p…i] $\leq$ pivot
   every element in A[j…r] $\geq$ pivot



Hoare's Partitioning Algorithm

1. Choose a pivot element: pivot = x = A[p]
2. Grow two regions:
   from left to right: A[p..i]
   from right to left: A[j..r]
   such that:
   every element in A[p…i] $\leq$ pivot
   every element in A[j…r] $\geq$ pivot



Hoare's Partitioning Algorithm

1. Choose a pivot element: pivot = x = A[p]
2. Grow two regions:

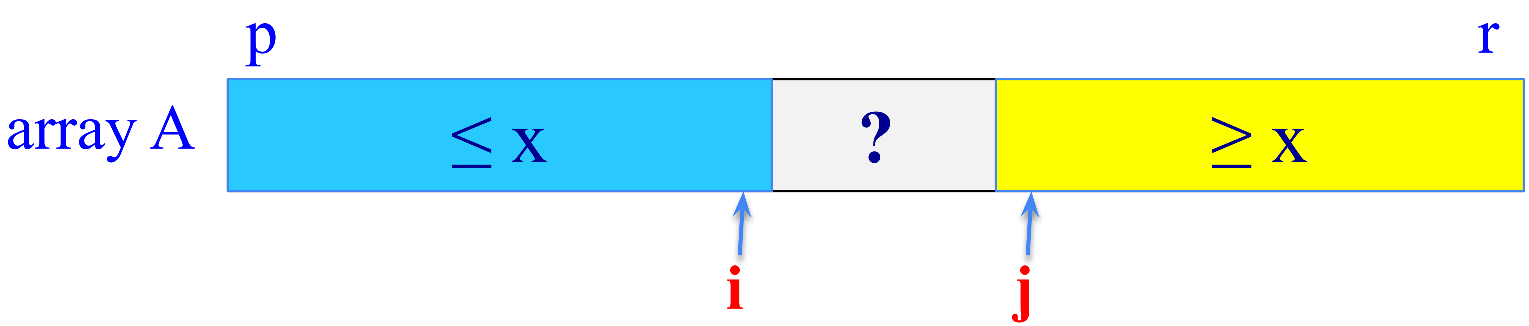
   from left to right: A[p..i]

   from right to left: A[j..r]

   such that:

   every element in A[p…i] $\leq$ pivot

   every element in A[j…r] $\geq$ pivot



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Partitioning Algorithm

1. Choose a pivot element: pivot = x = A[p]
2. Grow two regions:

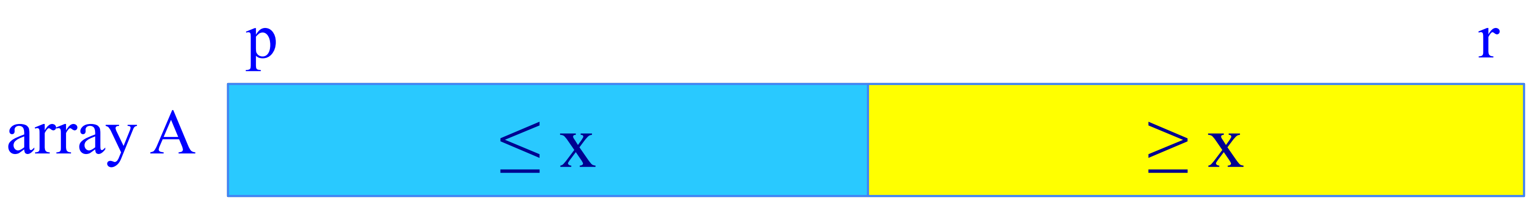
   from left to right: A[p..i]

   from right to left: A[j..r]

   such that:

   every element in A[p…i] $\leq$ pivot

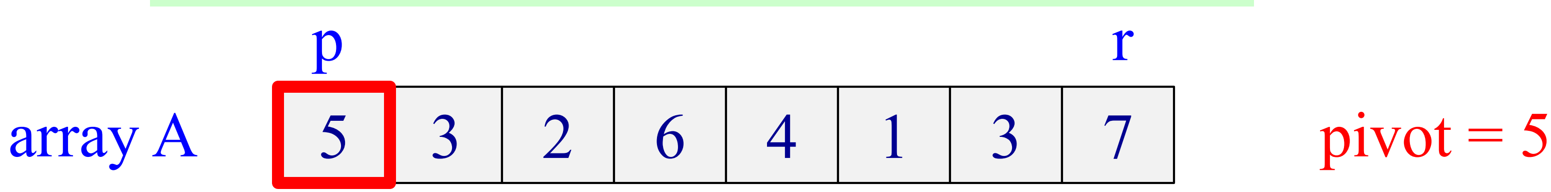
   every element in A[j…r] $\geq$ pivot



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Partitioning Algorithm

```
H-PARTITION (A, p, r)
    pivot ← A[p]
    i ← p − 1
    j ← r + 1
    while true do
        repeat j ← j − 1 until A[j] ≤ pivot
        repeat i ← i + 1 until A[i] ≥ pivot
        if i < j then exchange A[i] ↔ A[j]
        else return j
```



Hoare's Partitioning Algorithm

<u>H-PARTITION (A, *p*, *r*)</u>

pivot ← A[*p*]

i ← *p* − 1

j ← *r* + 1

while **true** **do**

repeat *j* ← *j* − 1 **until** A[*j*] ≤ *pivot*

repeat *i* ← *i* + 1 **until** A[*i*] ≥ *pivot*

if *i* < *j* **then** exchange A[*i*] ↔ A[*j*]

else return *j*

	p							r	
array A	5	3	2	6	4	1	3	7	pivot = 5

i (before p), j (after r)

Hoare's Partitioning Algorithm

<u>H-PARTITION (A, *p*, *r*)</u>

pivot ← A[*p*]

i ← *p* − 1

j ← *r* + 1

while true do

repeat *j* ← *j* − 1 **until** A[*j*] ≤ *pivot*

repeat *i* ← *i* + 1 **until** A[*i*] ≥ *pivot*

if *i* < *j* **then** exchange A[*i*] ↔ A[*j*]

else return *j*

p (first element), r (last element); i before the first element, j after the last element

5	3	2	6	4	1	3	7

array A

pivot = 5

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Partitioning Algorithm

```
H-PARTITION (A, p, r)
    pivot ← A[p]
    i ← p − 1
    j ← r + 1
    while true do
        repeat j ← j − 1 until A[j] ≤ pivot
        repeat i ← i + 1 until A[i] ≥ pivot
        if i < j then exchange A[i] ↔ A[j]
        else return j
```

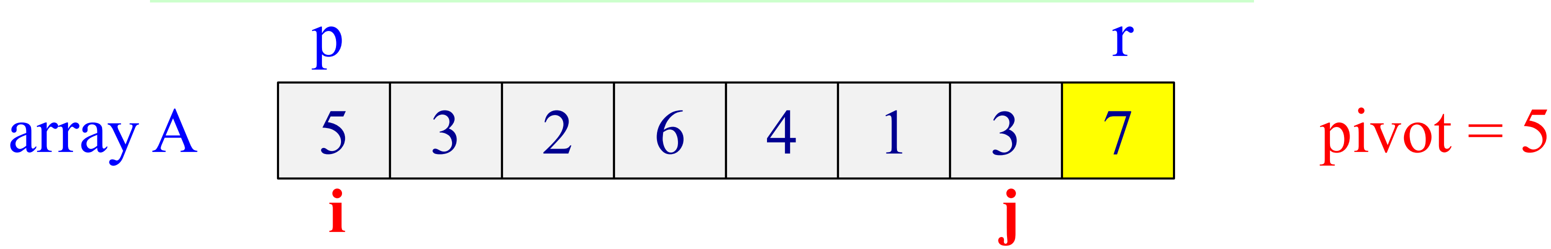
array A (p at first cell, r at last cell; i before first cell, j at seventh cell)

5	3	2	6	4	1	3	7

pivot = 5

Hoare's Partitioning Algorithm

```
H-PARTITION (A, p, r)
    pivot ← A[p]
    i ← p − 1
    j ← r + 1
    while true do
        repeat j ← j − 1 until A[j] ≤ pivot
        repeat i ← i + 1 until A[i] ≥ pivot
        if i < j then exchange A[i] ↔ A[j]
        else return j
```



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Partitioning Algorithm

```
H-PARTITION (A, p, r)
    pivot ← A[p]
    i ← p − 1
    j ← r + 1
    while true do
        repeat j ← j − 1 until A[j] ≤ pivot
        repeat i ← i + 1 until A[i] ≥ pivot
        if i < j then exchange A[i] ↔ A[j]
        else return j
```

	p							r
array A	3	3	2	6	4	1	5	7
	i						j	

pivot = 5

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Partitioning Algorithm

```
H-PARTITION (A, p, r)
    pivot ← A[p]
    i ← p − 1
    j ← r + 1
    while true do
        repeat j ← j − 1 until A[j] ≤ pivot
        repeat i ← i + 1 until A[i] ≥ pivot
        if i < j then exchange A[i] ↔ A[j]
        else return j
```

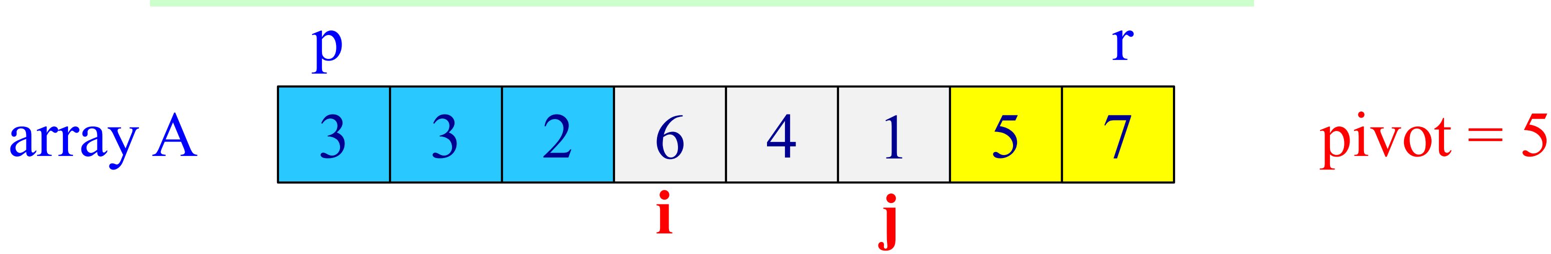
	p							r
array A	3	3	2	6	4	1	5	7
	i					j		

pivot = 5

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Partitioning Algorithm

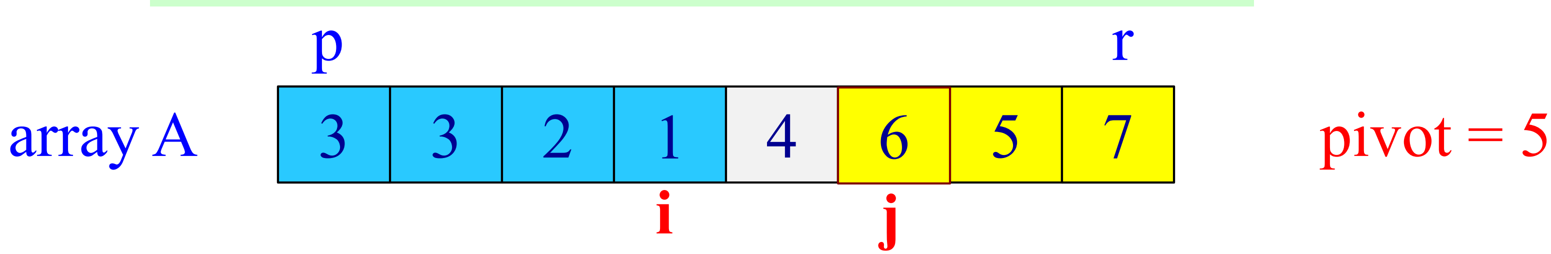
```
H-PARTITION (A, p, r)
    pivot ← A[p]
    i ← p − 1
    j ← r + 1
    while true do
        repeat j ← j − 1 until A[j] ≤ pivot
        repeat i ← i + 1 until A[i] ≥ pivot
        if i < j then exchange A[i] ↔ A[j]
        else return j
```



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Partitioning Algorithm

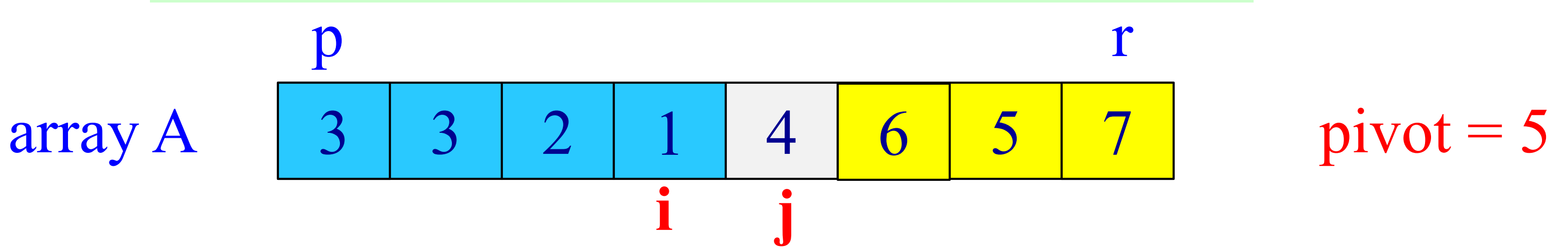
```
H-PARTITION (A, p, r)
    pivot ← A[p]
    i ← p − 1
    j ← r + 1
    while true do
        repeat j ← j − 1 until A[j] ≤ pivot
        repeat i ← i + 1 until A[i] ≥ pivot
        if i < j then exchange A[i] ↔ A[j]
        else return j
```



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Partitioning Algorithm

```
H-PARTITION (A, p, r)
    pivot ← A[p]
    i ← p − 1
    j ← r + 1
    while true do
        repeat j ← j − 1 until A[j] ≤ pivot
        repeat i ← i + 1 until A[i] ≥ pivot
        if i < j then exchange A[i] ↔ A[j]
        else return j
```



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Partitioning Algorithm

H-PARTITION (A, *p*, *r*)

pivot ← A[*p*]

i ← *p* − 1

j ← *r* + 1

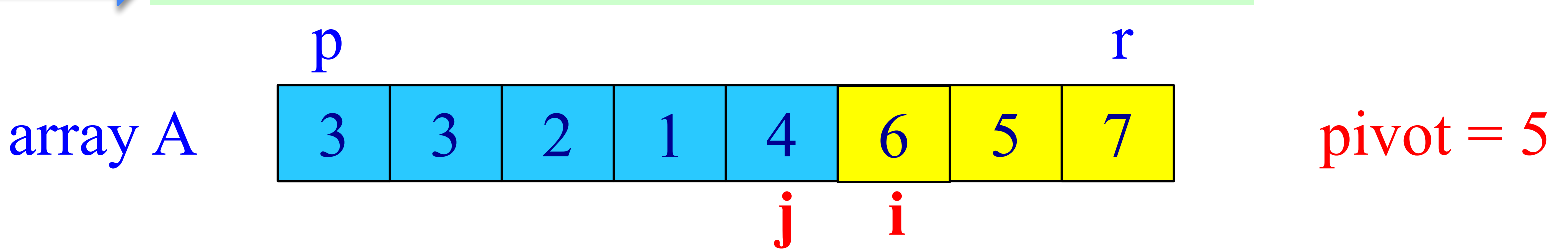
while **true** **do**

  repeat *j* ← *j* − 1 **until** A[*j*] ≤ *pivot*

  repeat *i* ← *i* + 1 **until** A[*i*] ≥ *pivot*

  if *i* < *j* **then** exchange A[*i*] ↔ A[*j*]

  else return *j*



Hoare's Partitioning Algorithm - Notes

<u>H-PARTITION (A, *p*, *r*)</u>

    pivot ← A[*p*]
    i ← *p* − 1
    j ← *r* + 1
    while true **do**
        repeat *j* ← *j* − 1 **until** A[*j*] ≤ *pivot*
        repeat *i* ← *i* + 1 **until** A[*i*] ≥ *pivot*
        if *i* < *j* **then** exchange A[*i*] ↔ A[*j*]
        else return *j*

Elements are exchanged when

- A[i] is **too large** to belong to the left region
- A[j] is **too small** to belong to the right region

assuming that the inequality is strict

The two regions A[p..i] and A[j..r] grow until

$A[i] \geq pivot \geq A[j]$

Hoare's Partitioning Algorithm

```
H-PARTITION (A, p, r)
    pivot ← A[p]
    i ← p − 1
    j ← r + 1
    while true do
        repeat j ← j − 1 until A[j] ≤ pivot
        repeat i ← i + 1 until A[i] ≥ pivot
        if i < j then exchange A[i] ↔ A[j]
        else return j
```

What is the asymptotic runtime of Hoare's partitioning algorithm?

$\Theta(n)$

QUICKSORT (A, p, r)

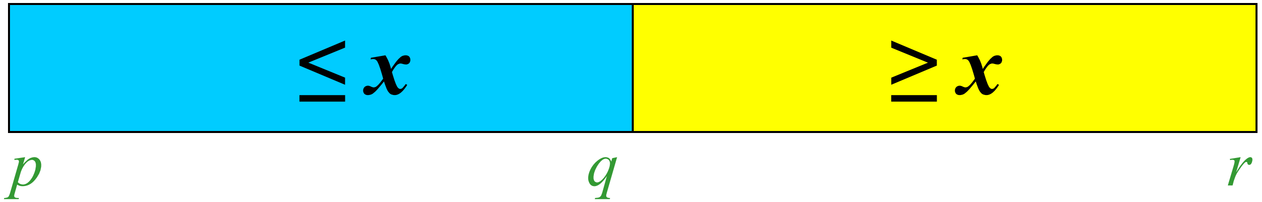
if $p < r$ then

$q \leftarrow$ H-PARTITION(A, p, r)

QUICKSORT(A, p, q)

QUICKSORT(A, q +1, r)

Initial invocation: QUICKSORT(A, 1, n)



Question

H-PARTITION (A, p, r)

```
pivot ← A[p]
i ← p − 1
j ← r + 1
while true do
    repeat j ← j − 1 until A[j] ≤ pivot
    repeat i ← i + 1 until A[i] ≥ pivot
    if i < j then exchange A[i] ↔ A[j]
    else return j
```

QUICKSORT (A, p, r)

```
if p < r then
    q ← H-PARTITION(A, p, r)
    QUICKSORT(A, p, q)
    QUICKSORT(A, q +1, r)
```

Q: What happens if we select pivot to be A[r] instead of A[p] in H-PARTITION?

✖ a) *QUICKSORT* will still work correctly.

✖ b) *QUICKSORT* may return incorrect results for some inputs.

✔ c) *QUICKSORT* may not terminate for some inputs.

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Partitioning Algorithm: Pivot Selection

H-PARTITION (A, p, r)

- $pivot \leftarrow A[p]$
- $i \leftarrow p - 1$
- $j \leftarrow r + 1$
- **while true do**
  - **repeat** $j \leftarrow j - 1$ **until** $A[j] \leq pivot$
  - **repeat** $i \leftarrow i + 1$ **until** $A[i] \geq pivot$
  - **if** $i < j$ **then** exchange $A[i] \leftrightarrow A[j]$
  - **else** return j

QUICKSORT (A, p, r)

- if $p < r$ then
  - $q \leftarrow$ H-PARTITION(A, p, r)
  - QUICKSORT(A, p, q)
  - QUICKSORT(A, q +1, r)

If A[r] is chosen as the pivot:

Consider the example where A[r] is the largest element in the array:

5	3	6	4	3	**7**

End of H-PARTITION: **i = j = r**

In QUICKSORT: **q = r**

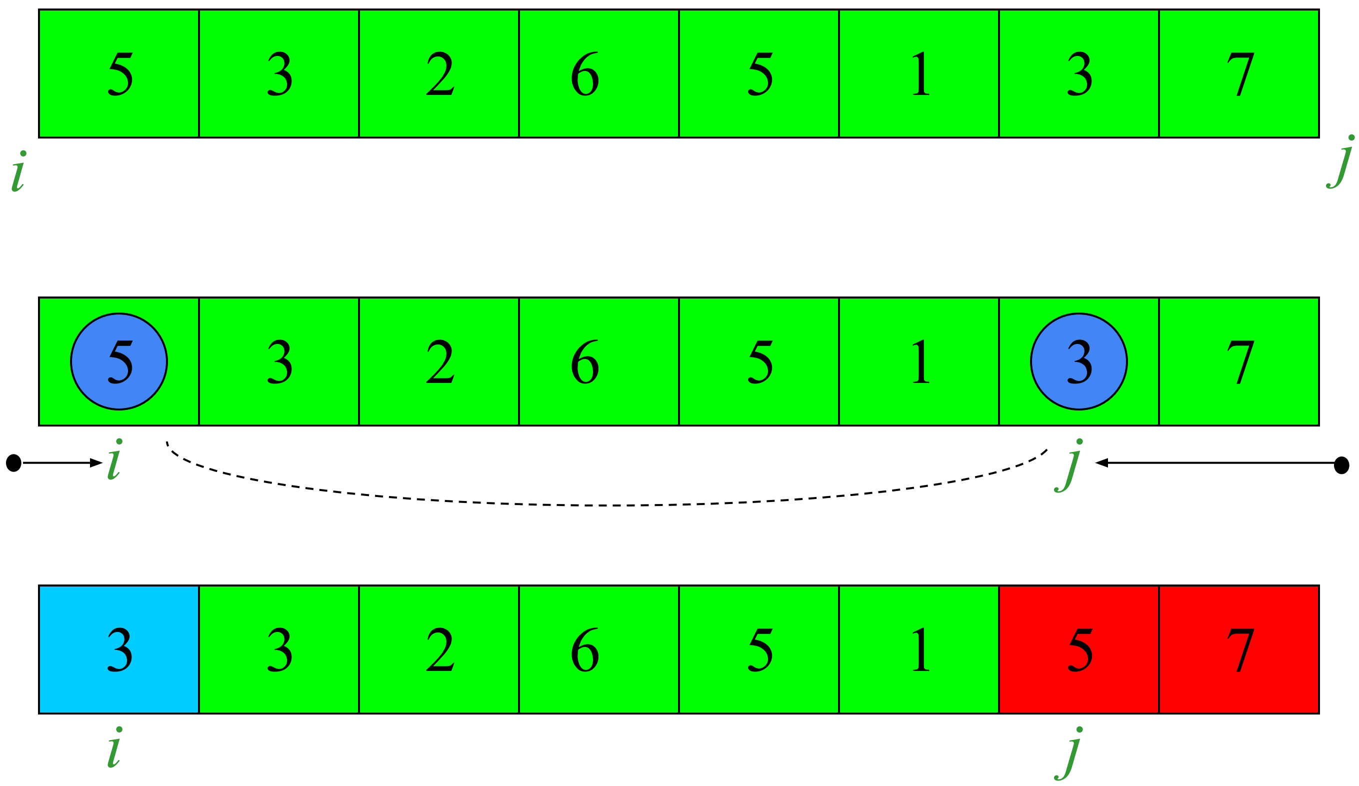
So, recursive call to:

QUICKSORT (A, p, q=r)

□ **infinite loop**

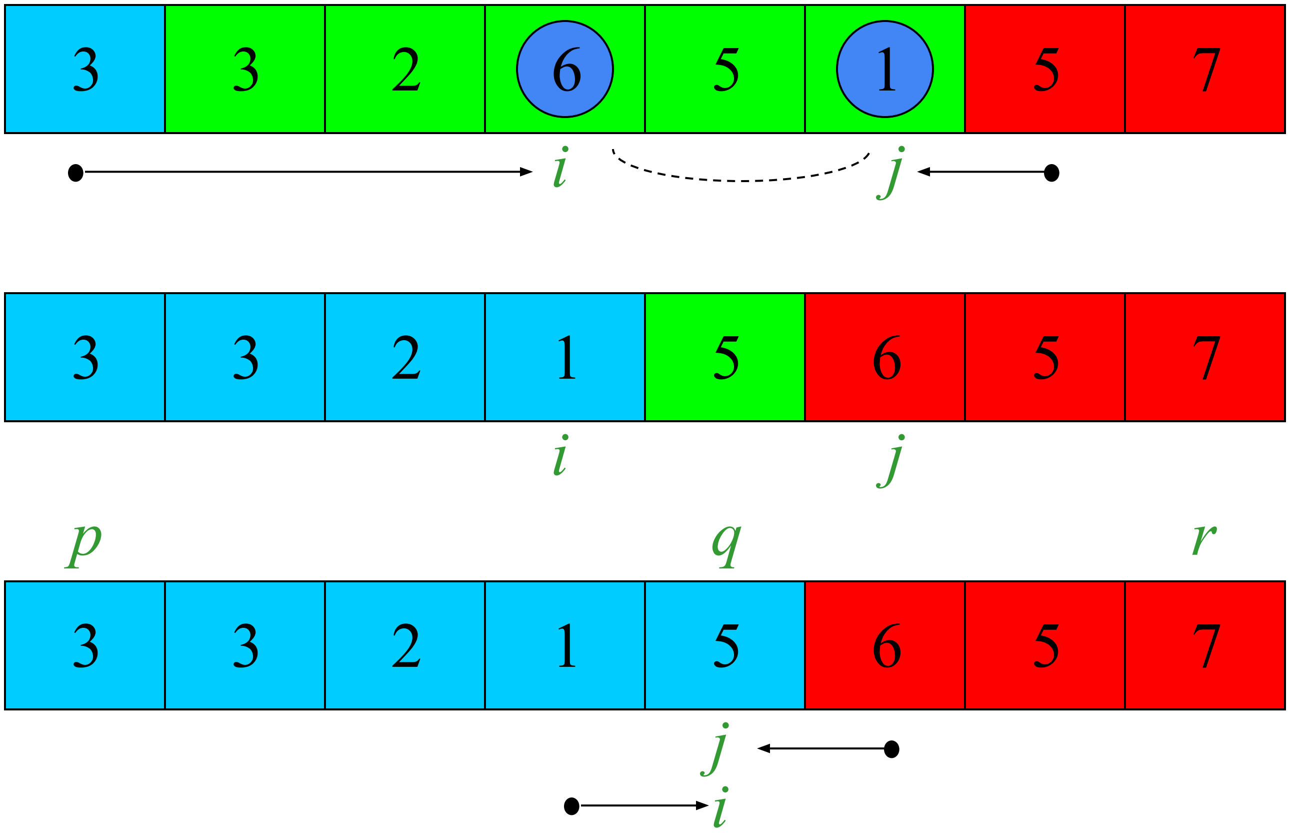
Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Algorithm: Example 2 (pivot = 5)



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Hoare's Algorithm: Example 2 (pivot = 5)



Termination: $i = j = 5$

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

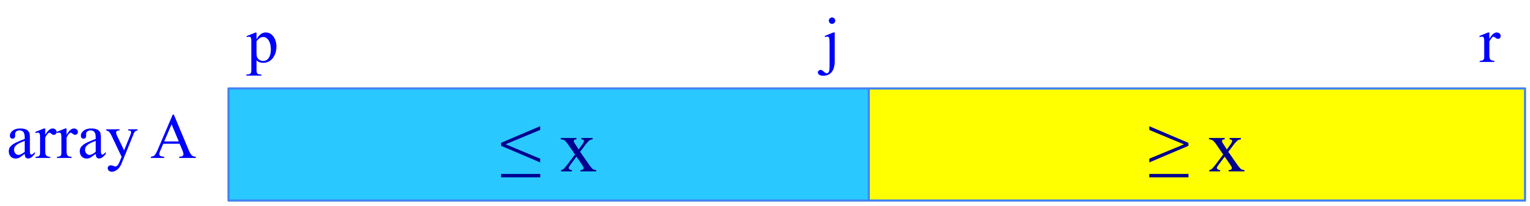
Correctness of Hoare's Algorithm

We need to prove 3 claims to show correctness:

a) Indices i & j never reference A outside the interval $A[p..r]$

b) Split is always non-trivial; i.e., $j \neq r$ at termination

c) Every element in $A[p..j] \leq$ every element in $A[j+1..r]$ at termination



Correctness of Hoare's Algorithm

Notations:

k: # of times the while-loop iterates until termination

i_m: the value of index i at the end of iteration m

j_m: the value of index j at the end of iteration m

x: the value of the pivot element

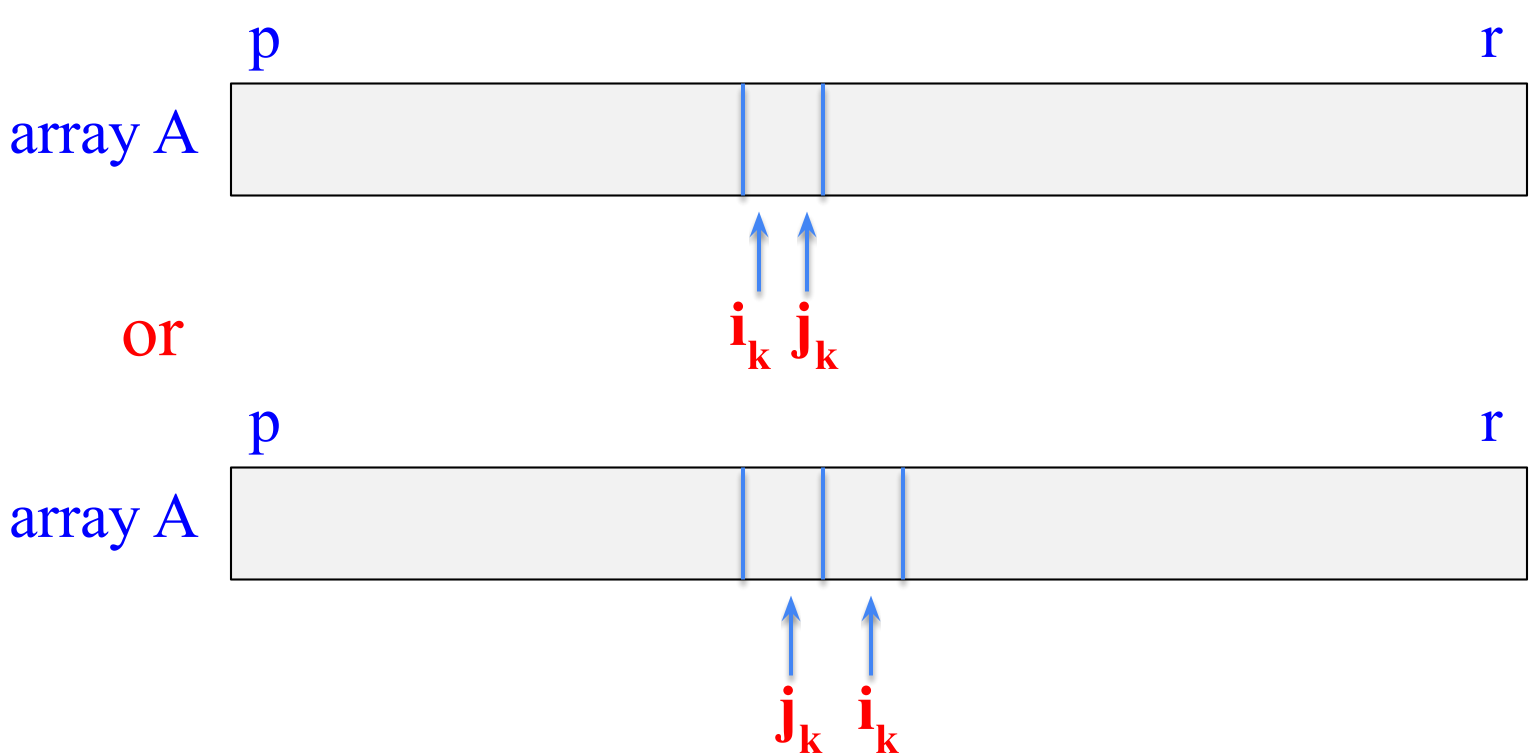
Note: We always have $i_1 = p$ and $p \le j_1 \le r$

because $x = A[p]$

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University
Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Correctness of Hoare's Algorithm

Lemma 1: Either $i_k = j_k$ or $i_k = j_k + 1$ at termination



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Correctness of Hoare's Algorithm

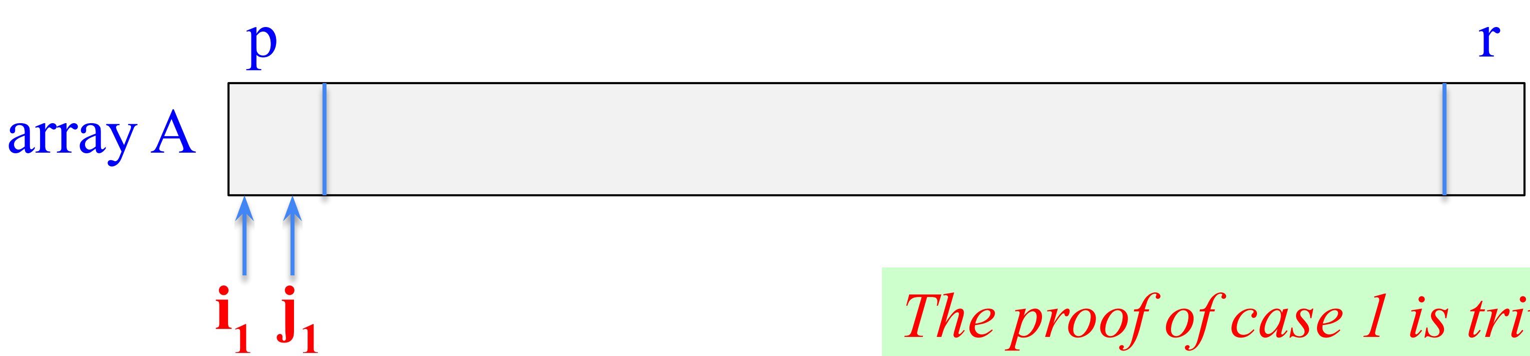
Proof of Lemma 1:

The algorithm terminates when $i \geq j$ (the else condition).

So, it is sufficient to prove that $\boldsymbol{i_k - j_k \leq 1}$

There are 2 cases to consider:

Case 1: $k = 1$, i.e. the algorithm terminates in a single iteration



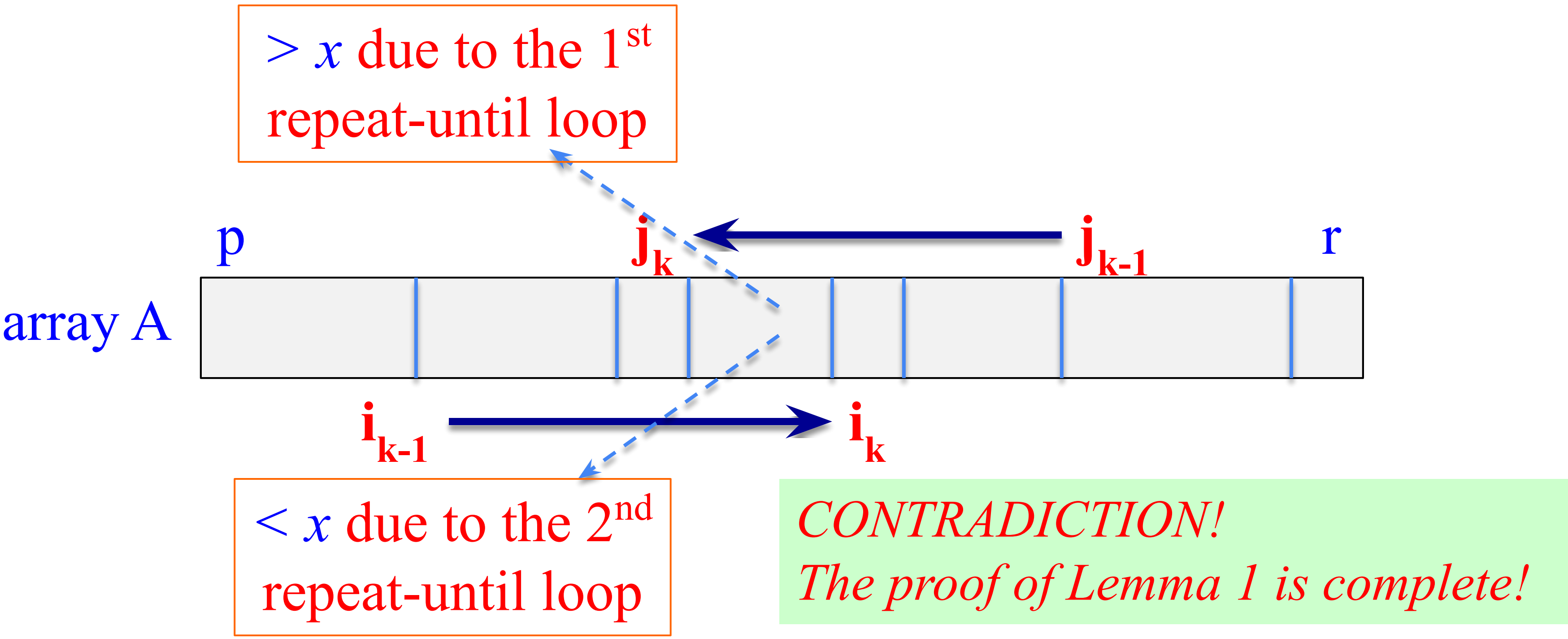
The proof of case 1 is trivial

Correctness of Hoare's Algorithm

Proof of Lemma 1 (cont'd):

Case 2: $k > 1$, i.e. the alg. does not terminate in a single iter.

By contradiction, assume there is a run with $i_k - j_k > 1$



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Correctness of Hoare's Algorithm

Original correctness claims:

(a) Indices *i* & *j* never reference A outside the interval A[*p*…*r*]

(b) Split is always non-trivial; i.e., $j \neq r$ at termination

Proof:

For k = 1: Trivial because $i_1 = j_1 = p$ *(see Case 1 in proof of Lemma 2)*

For k > 1:

$i_k > p$ and $j_k < r$ *(due to the repeat-until loops moving indices)*

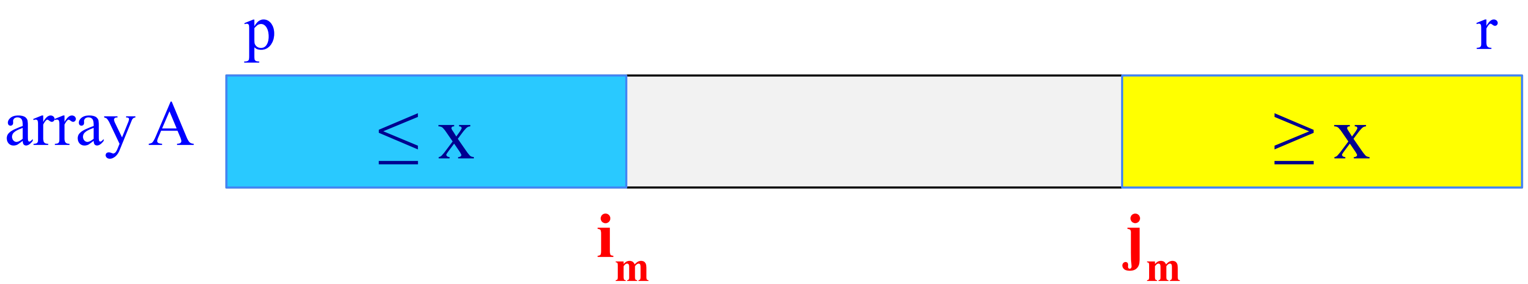
$i_k \leq r$ and $j_k \geq p$ *(due to Lemma 1 and the statement above)*

□ The proof of claims (a) and (b) complete

Correctness of Hoare's Algorithm

Lemma 2: At the end of iteration m, where $m < k$ *(i.e. m is not the last iteration)*, we must have:

$A[p..i_m] \leq x$ and $A[j_m .. r] \geq x$

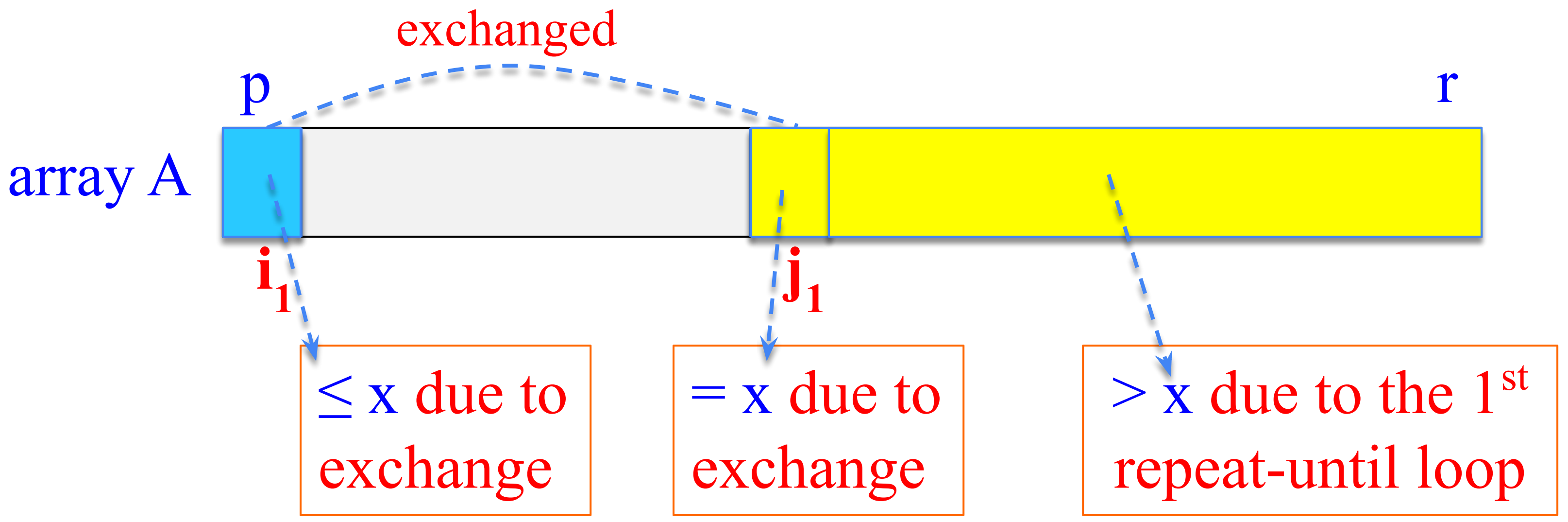


Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Correctness of Hoare's Algorithm

Proof of Lemma 2:

Base case: $m=1$ and $k > 1$ *(i.e. the alg. does not terminate in the first iter.)*



Proof of base case complete!

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Correctness of Hoare's Algorithm

Proof of Lemma 2 (cont'd):

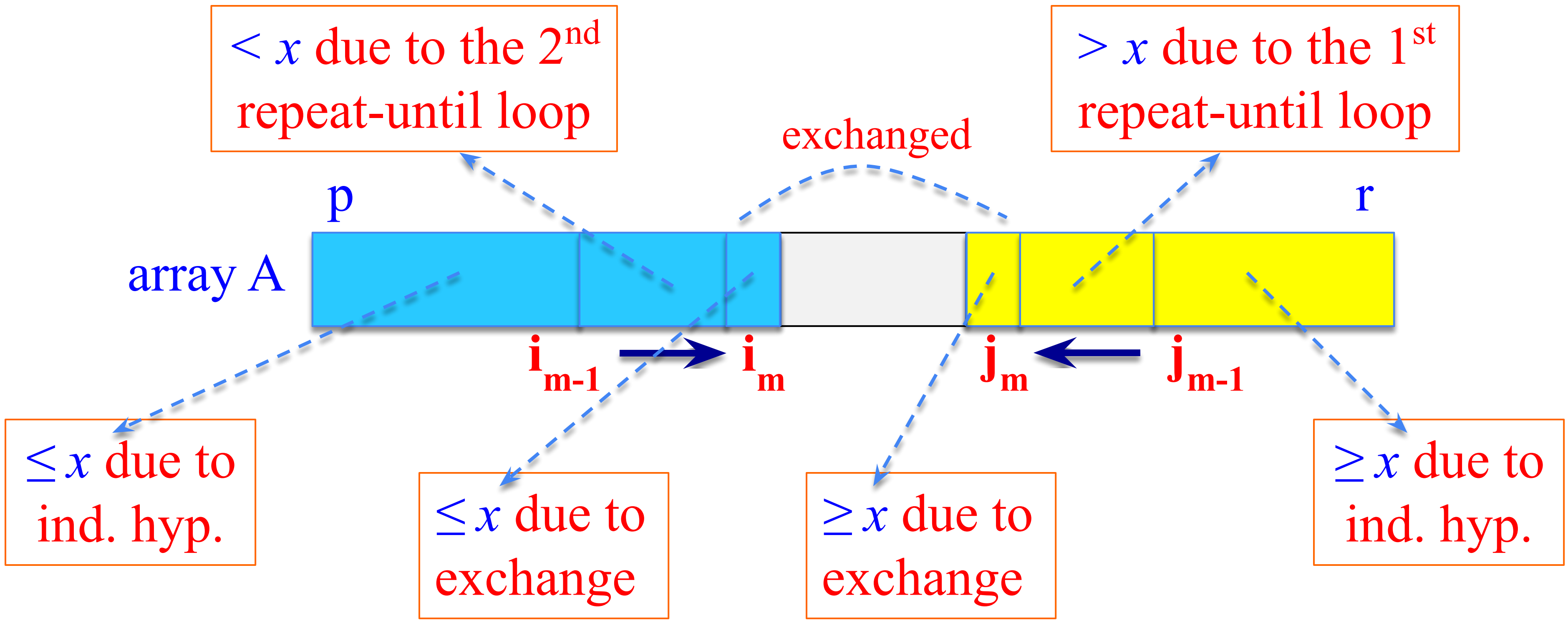
Inductive hypothesis: At the end of iteration $m\text{-}1$, where $m < k$ *(i.e. m is not the last iteration)*, we must have:

$$A[p..i_{m-1}] \leq x \quad \text{and} \quad A[j_{m-1}..r] \geq x$$

General case: The lemma holds for m, where $m < k$

Correctness of Hoare's Algorithm

For $1 < m < k$, at the end of iteration m, we have:



Proof of Lemma 2 complete!

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Correctness of Hoare's Algorithm

Original correctness claim:

(c) Every element in $A[p \ldots j] \leq$ every element in $A[j+1 \ldots r]$ at termination

Proof of claim (c)

There are 3 cases to consider:

Case 1: $k = 1$, *i.e. the algorithm terminates in a single iteration*

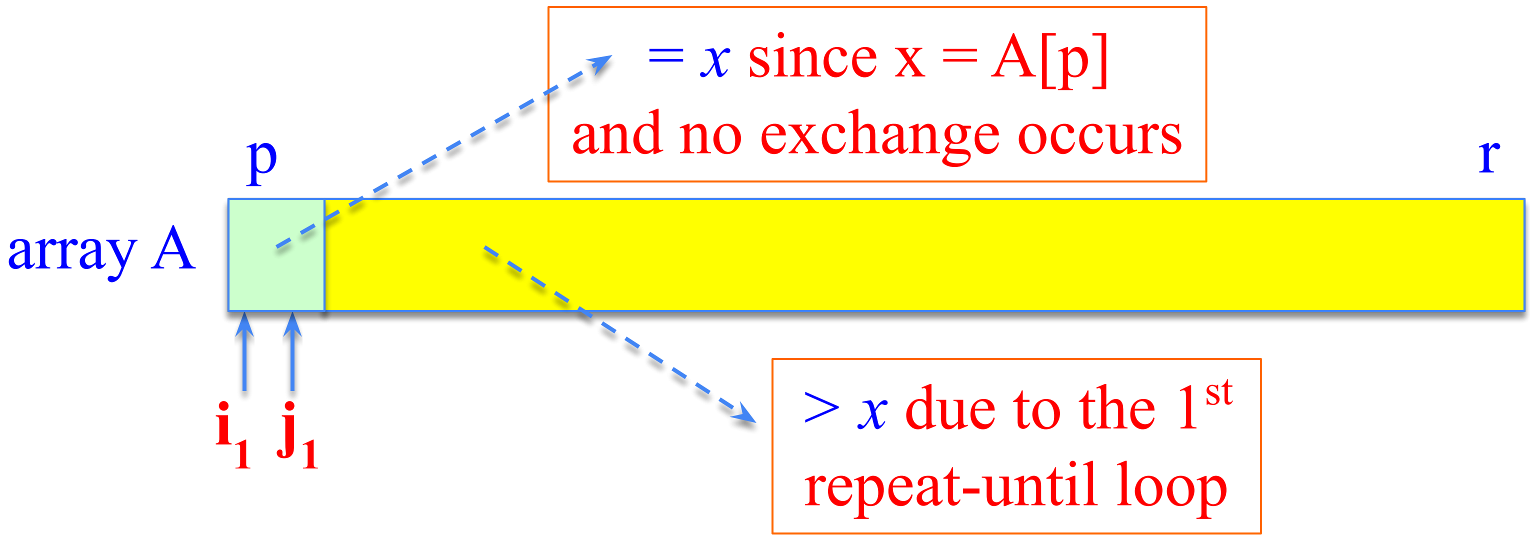
Case 2: $k > 1$ and $i_k = j_k$

Case 3: $k > 1$ and $i_k = j_k + 1$

Correctness of Hoare's Algorithm

<u>Proof of claim (c):</u>

<u>Case 1</u>: $k = 1$, *i.e. the algorithm terminates in a single iteration*

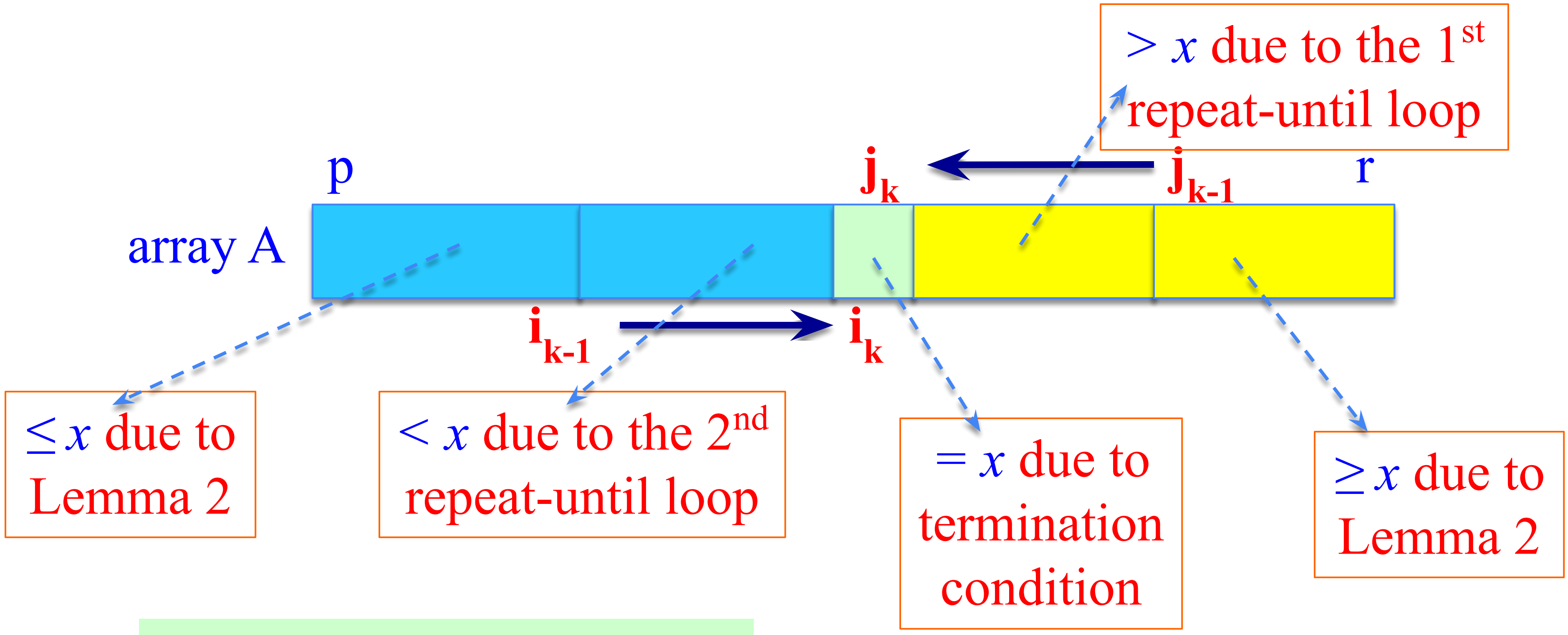


Proof of case 1 complete!

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Correctness of Hoare's Algorithm

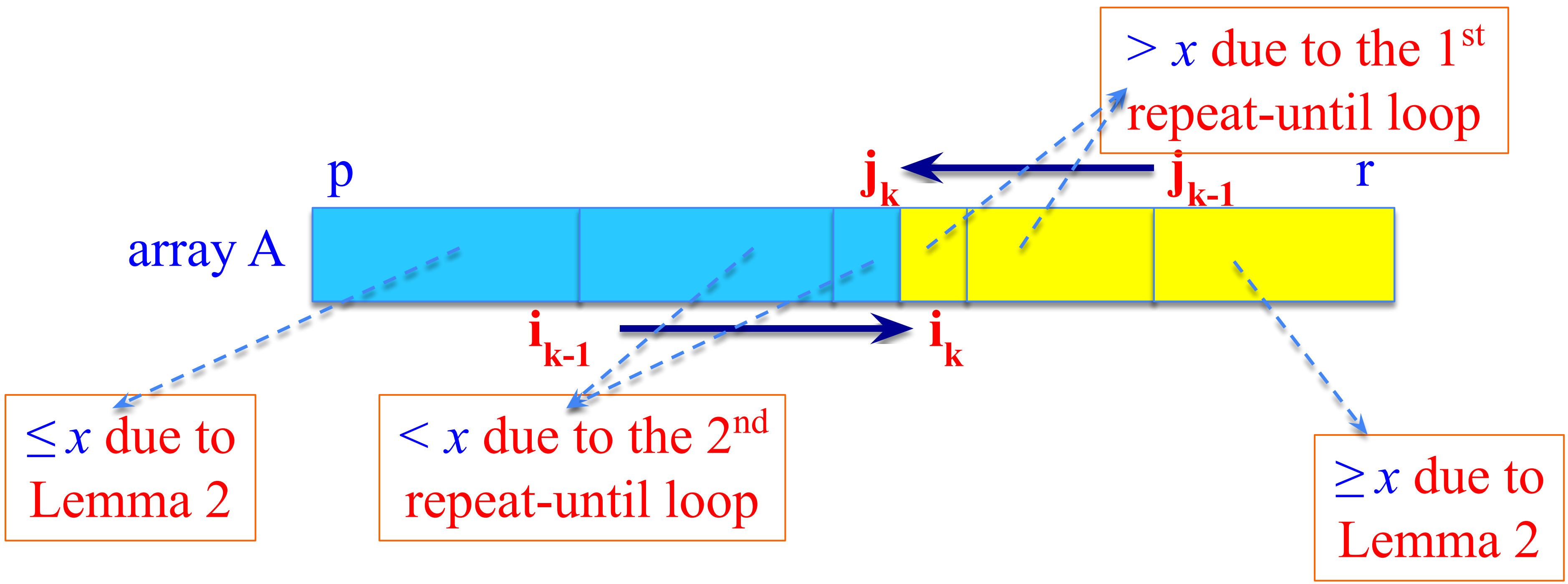
Proof of claim (c) (cont'd): Case 2: $k > 1$ and $i_k = j_k$



Proof of Case 2 complete!

Correctness of Hoare's Algorithm

Proof of claim (c) (cont'd): Case 3: $k > 1$ and $i_k = j_k + 1$



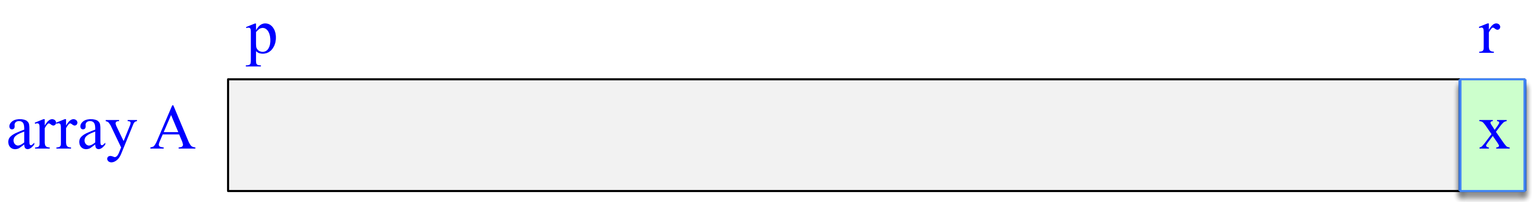
Proof of Case 3 complete!

Correctness proof complete!

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Lomuto's Partitioning Algorithm

1. Choose a pivot element: $pivot = x = A[r]$
2. Grow two regions:
   from left to right: $A[p..i]$
   from left to right: $A[i+1..j]$
   such that:
   every element in $A[p \ldots i] \leq pivot$
   every element in $A[i+1 \ldots j] > pivot$



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Lomuto's Partitioning Algorithm

1. Choose a pivot element: $pivot = x = A[r]$
2. Grow two regions:

   from left to right: $A[p..i]$

   from left to right: $A[i+1..j]$

   such that:

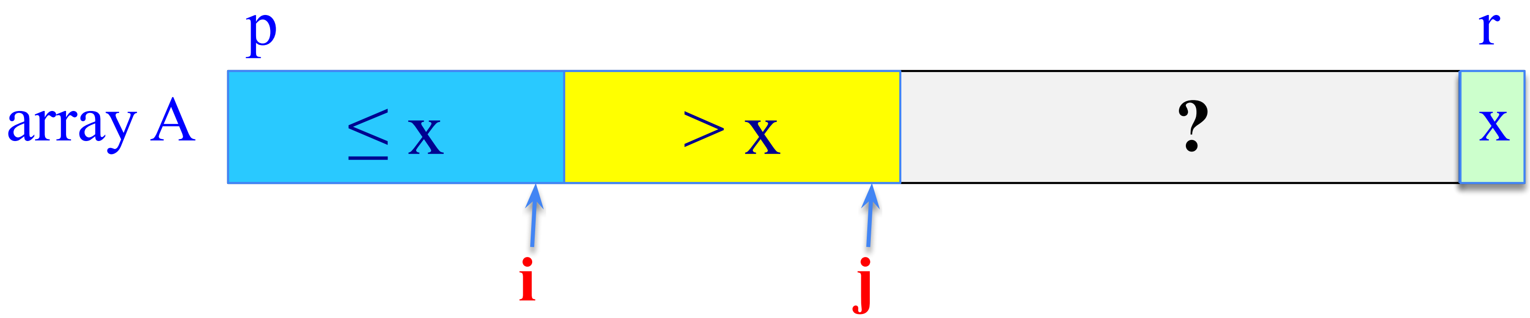
   every element in $A[p \ldots i] \leq pivot$

   every element in $A[i+1 \ldots j] > pivot$

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

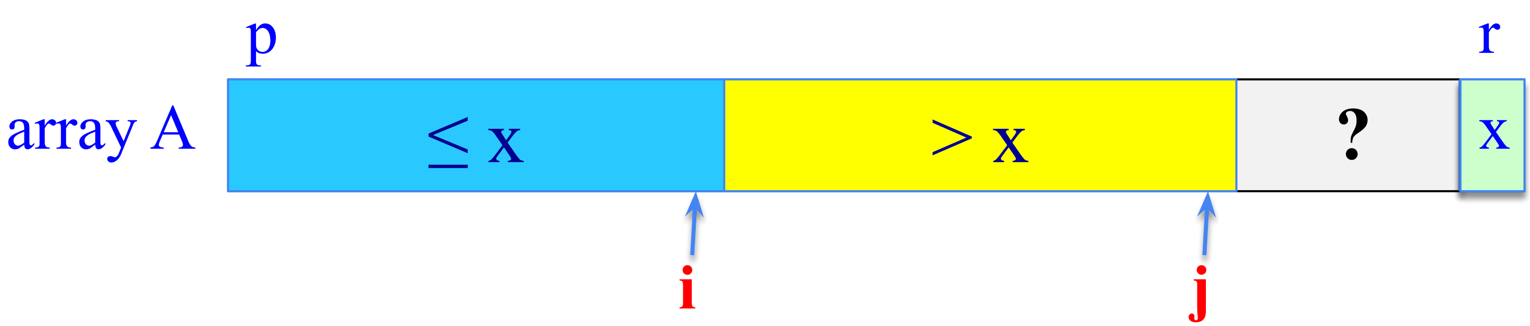
Lomuto's Partitioning Algorithm

1. Choose a pivot element: $pivot = x = A[r]$
2. Grow two regions:
   from left to right: $A[p..i]$
   from left to right: $A[i+1..j]$
   such that:
   every element in $A[p \ldots i] \leq pivot$
   every element in $A[i+1 \ldots j] > pivot$



Lomuto's Partitioning Algorithm

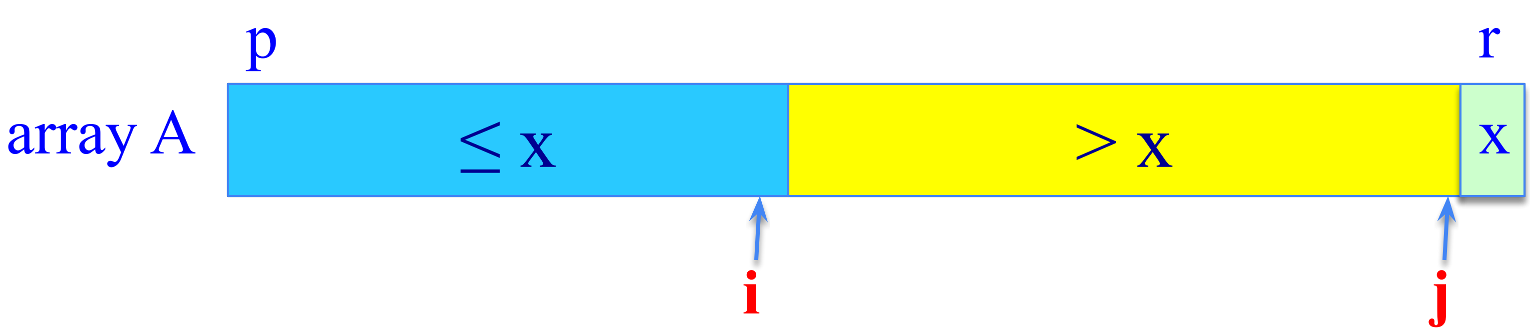
1. Choose a pivot element: *pivot = x = A[r]*
2. Grow two regions:
   from left to right: *A[p..i]*
   from left to right: *A[i+1..j]*
   such that:
   every element in $A[p \ldots i] \leq pivot$
   every element in $A[i+1 \ldots j] > pivot$



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

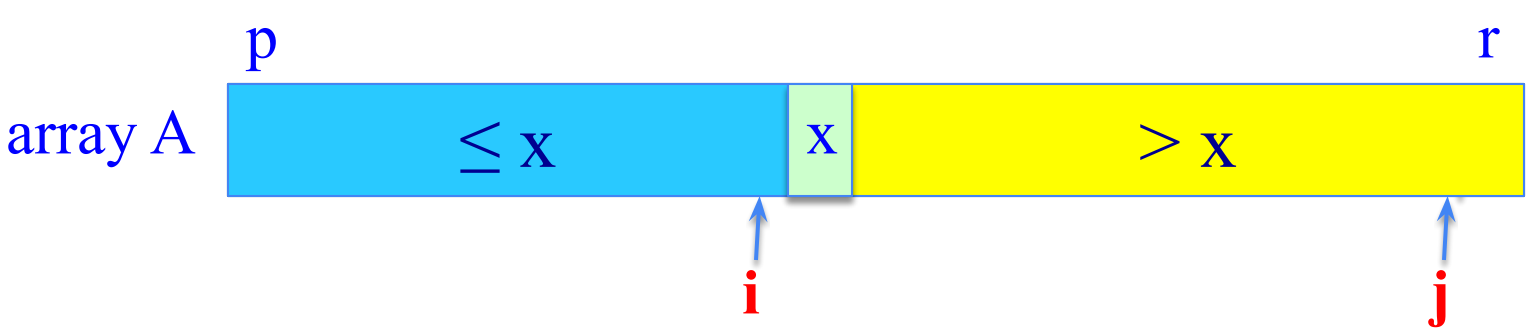
Lomuto's Partitioning Algorithm

1. Choose a pivot element: $pivot = x = A[r]$
2. Grow two regions:
   from left to right: $A[p..i]$
   from left to right: $A[i+1..j]$
   such that:
   every element in $A[p \ldots i] \leq pivot$
   every element in $A[i+1 \ldots j] > pivot$



Lomuto's Partitioning Algorithm

1. Choose a pivot element: $pivot = x = A[r]$
2. Grow two regions:
   from left to right: $A[p..i]$
   from left to right: $A[i+1..j]$
   such that:
   every element in $A[p\ldots i] \le pivot$
   every element in $A[i+1\ldots j] > pivot$



Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

```
    pivot ← A[r]
    i ← p − 1
    for j ← p to r − 1 do
        if A[j] ≤ pivot then
            i ← i + 1
            exchange A[i] ↔ A[j]
    exchange A[i + 1] ↔ A[r]
    return i + 1
```

	p							r
array A	7	8	2	6	5	1	3	4

pivot = 4

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

pivot ← A[*r*]
i ← *p* − 1
for *j* ← *p* **to** *r* − 1 **do**
    if A[*j*] ≤ *pivot* **then**
        i ← *i* + 1
        exchange A[*i*] ↔ A[*j*]
exchange A[*i* + 1] ↔ A[*r*]
return *i* + 1

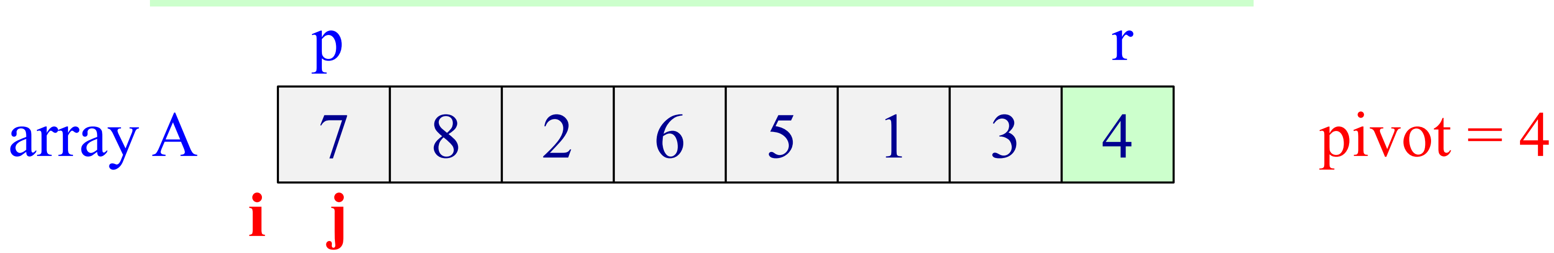
		p							r
array A		7	8	2	6	5	1	3	4
	i	j							

pivot = 4

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Lomuto's Partitioning Algorithm

```
L-PARTITION (A, p, r)
    pivot ← A[r]
    i ← p − 1
    for j ← p to r − 1 do
        if A[j] ≤ pivot then
            i ← i + 1
            exchange A[i] ↔ A[j]
    exchange A[i + 1] ↔ A[r]
    return i + 1
```



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

$pivot \leftarrow A[r]$

$i \leftarrow p - 1$

for $j \leftarrow p$ **to** $r - 1$ **do**

  if $A[j] \leq pivot$ **then**

    $i \leftarrow i + 1$

    exchange $A[i] \leftrightarrow A[j]$

exchange $A[i + 1] \leftrightarrow A[r]$

return $i + 1$

	p							r
array A	7	8	2	6	5	1	3	4

i (left of 7), j (at 2)

pivot = 4

Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

$pivot \leftarrow A[r]$

$i \leftarrow p - 1$

for $j \leftarrow p$ **to** $r - 1$ **do**

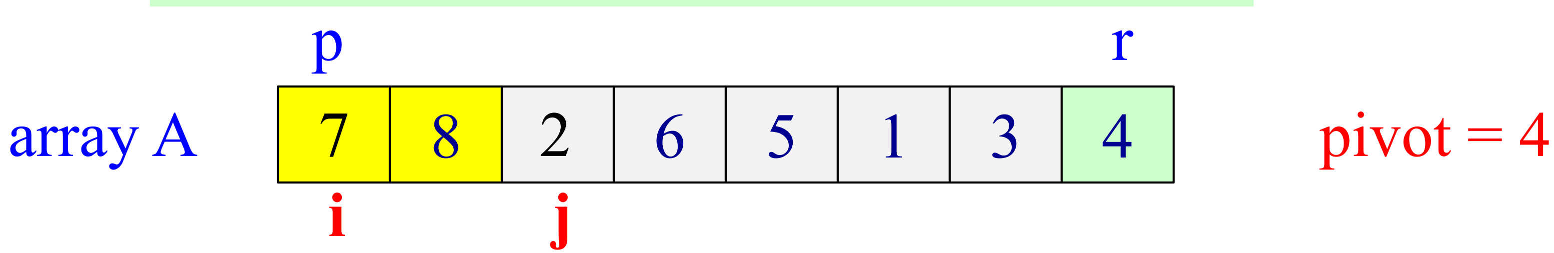
  if $A[j] \leq pivot$ **then**

    $i \leftarrow i + 1$

    exchange $A[i] \leftrightarrow A[j]$

exchange $A[i + 1] \leftrightarrow A[r]$

return $i + 1$



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

```
pivot ← A[r]
i ← p − 1
for j ← p to r − 1 do
    if A[j] ≤ pivot then
        i ← i + 1
        exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

	p							r	
array A	2	8	7	6	5	1	3	4	pivot = 4
	i		j						

Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

$pivot \leftarrow A[r]$
$i \leftarrow p - 1$
for $j \leftarrow p$ **to** $r - 1$ **do**
  if $A[j] \leq pivot$ **then**
    $i \leftarrow i + 1$
    exchange $A[i] \leftrightarrow A[j]$
exchange $A[i + 1] \leftrightarrow A[r]$
return $i + 1$

	p							r
array A	2	8	7	6	5	1	3	4
	i					j		

pivot = 4

Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

```
pivot ← A[r]
i ← p − 1
for j ← p to r − 1 do
    if A[j] ≤ pivot then
        i ← i + 1
        exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

	p							r	
array A	2	8	7	6	5	1	3	4	pivot = 4
		i				j			

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

```
pivot ← A[r]
i ← p − 1
for j ← p to r − 1 do
    if A[j] ≤ pivot then
        i ← i + 1
        exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

	p							r
array A	2	1	7	6	5	8	3	4
		i				j		

pivot = 4

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

```
pivot ← A[r]
i ← p − 1
for j ← p to r − 1 do
    if A[j] ≤ pivot then
        i ← i + 1
        exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

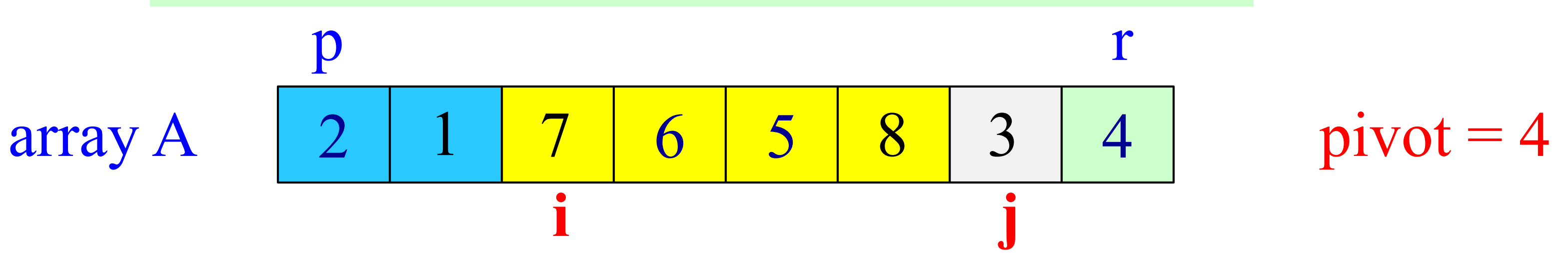
	p							r	
array A	2	1	7	6	5	8	3	4	pivot = 4
		i					j		

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

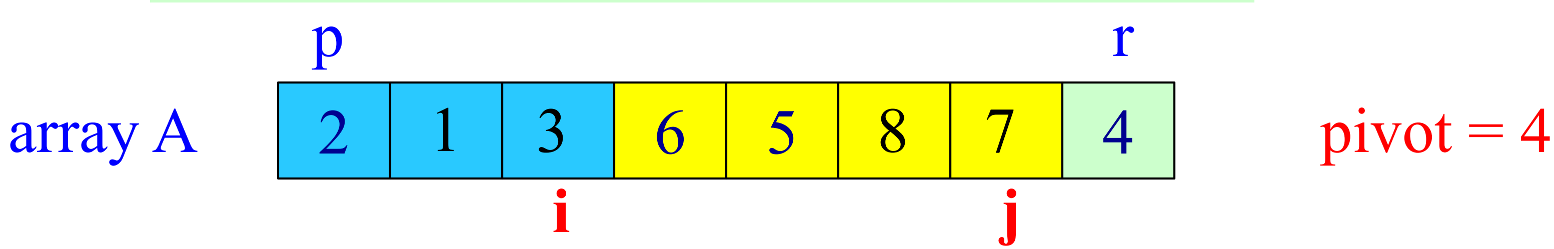
```
pivot ← A[r]
i ← p − 1
for j ← p to r − 1 do
    if A[j] ≤ pivot then
        i ← i + 1
        exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```



Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

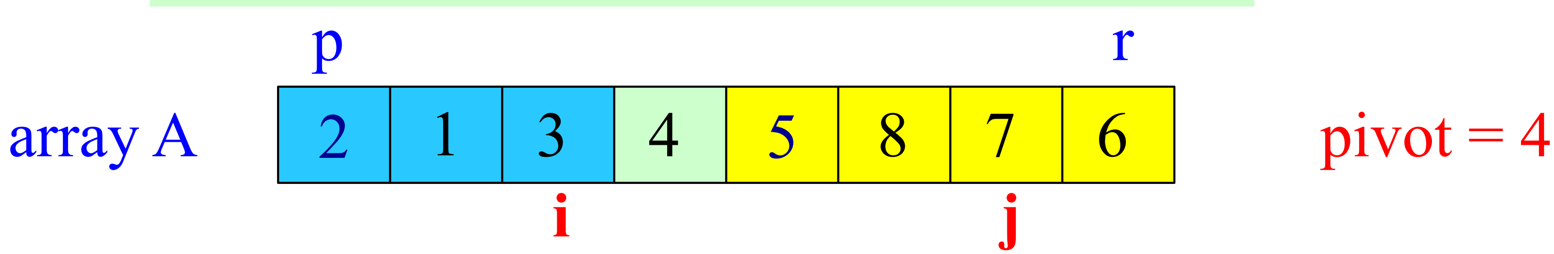
```
pivot ← A[r]
i ← p − 1
for j ← p to r − 1 do
    if A[j] ≤ pivot then
        i ← i + 1
        exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```



Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

```
pivot ← A[r]
i ← p − 1
for j ← p to r − 1 do
    if A[j] ≤ pivot then
        i ← i + 1
        exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

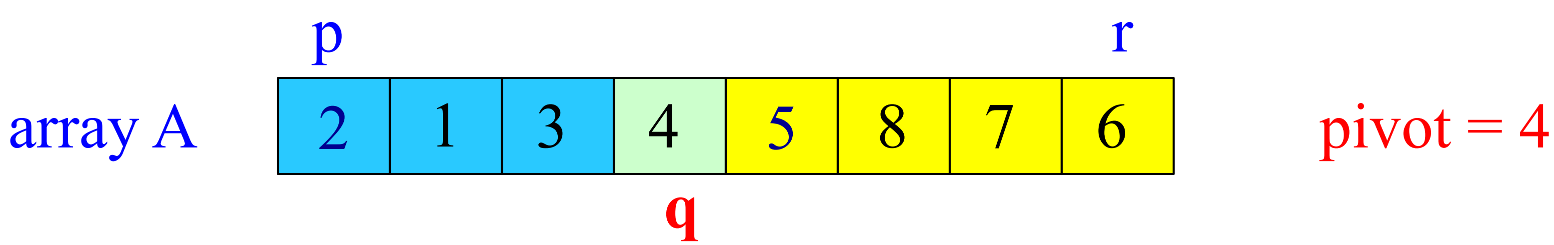


Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Lomuto's Partitioning Algorithm

L-PARTITION (A, *p*, *r*)

```
pivot ← A[r]
i ← p − 1
for j ← p to r − 1 do
    if A[j] ≤ pivot then
        i ← i + 1
        exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```



Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Lomuto's Partitioning Algorithm

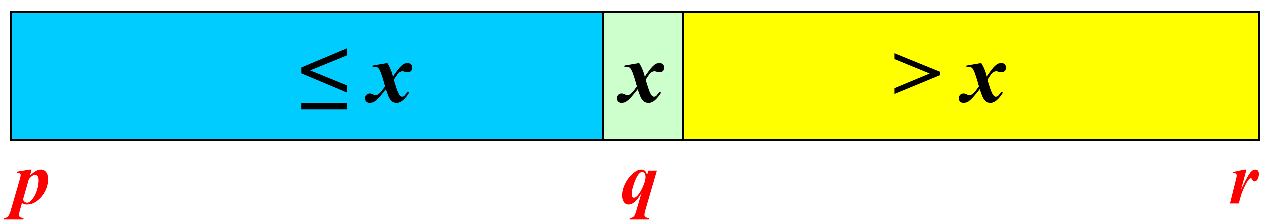
L-PARTITION (A, *p*, *r*)

$pivot \leftarrow A[r]$
$i \leftarrow p - 1$
for $j \leftarrow p$ **to** $r - 1$ **do**
  if $A[j] \leq pivot$ **then**
    $i \leftarrow i + 1$
    exchange $A[i] \leftrightarrow A[j]$
exchange $A[i + 1] \leftrightarrow A[r]$
return $i + 1$

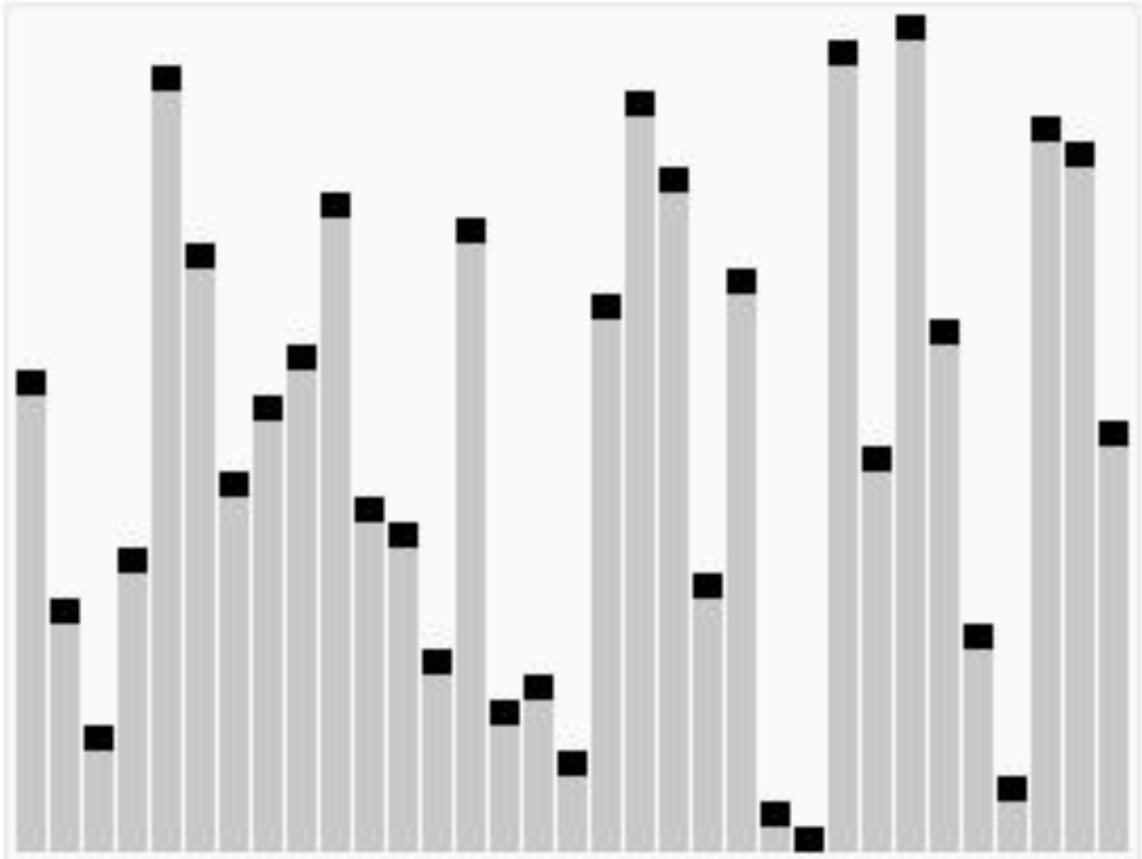
What is the runtime of L-PARTITION?   **Θ(n)**

```
QUICKSORT (A, p, r)
    if p < r then
        q ← L-PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q +1, r)
```

Initial invocation: QUICKSORT(A, 1, n)



Quicksort Animation



from Wikimedia Commons

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Comparison of Hoare's & Lomuto's Algorithms

Notation: $n = r - p + 1$ & $pivot = A[p]$ (Hoare)

& $pivot = A[r]$ (Lomuto)

- # of element exchanges: $e(n)$
  - Hoare: $0 \leq e(n) \leq$
    - Best: $k = 1$ with $i_1 = j_1 = p$ (i.e., $A[\, p+1 \ldots r\,] > pivot$)
    - Worst: $A[\, p+1 \ldots p+ \quad -1] \geq pivot \geq A[\, p+ \quad \ldots r\,]$
  - Lomuto: $1 \leq e(n) \leq n$
    - Best: $A[\, p \ldots r-1\,] > pivot$
    - Worst: $A[\, p \ldots r-1\,] \leq pivot$

Comparison of Hoare's & Lomuto's Algorithms

- # of element comparisons: $c_e(n)$
  - Hoare: $n + 1 \le c_e(n) \le n + 2$
    - Best: $i_k = j_k$
    - Worst: $i_k = j_k + 1$
  - Lomuto: $c_e(n) = n - 1$

- # of index comparisons: $c_i(n)$
  - Hoare: $1 \le c_i(n) \le \quad + 1 \quad (c_i(n) = e(n) + 1)$
  - Lomuto: $c_i(n) = n - 1$

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Comparison of Hoare's & Lomuto's Algorithms

- # of index increment/decrement operations: $a(n)$
  - Hoare: $n + 1 \leq a(n) \leq n + 2 \quad (a(n) = c_e(n))$
  - Lomuto: $n \leq a(n) \leq 2n - 1 \quad (a(n) = e(n) + (n - 1))$

- Hoare's algorithm is in general faster

- Hoare behaves better when pivot is repeated in A[$p \ldots r$]
  - Hoare: Evenly distributes them between left & right regions
  - Lomuto: Puts all of them to the left region

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University