ALGORITHMS

(CSCI 2200)

Week 4

Quicksort

Laura Toma

Bowdoin College

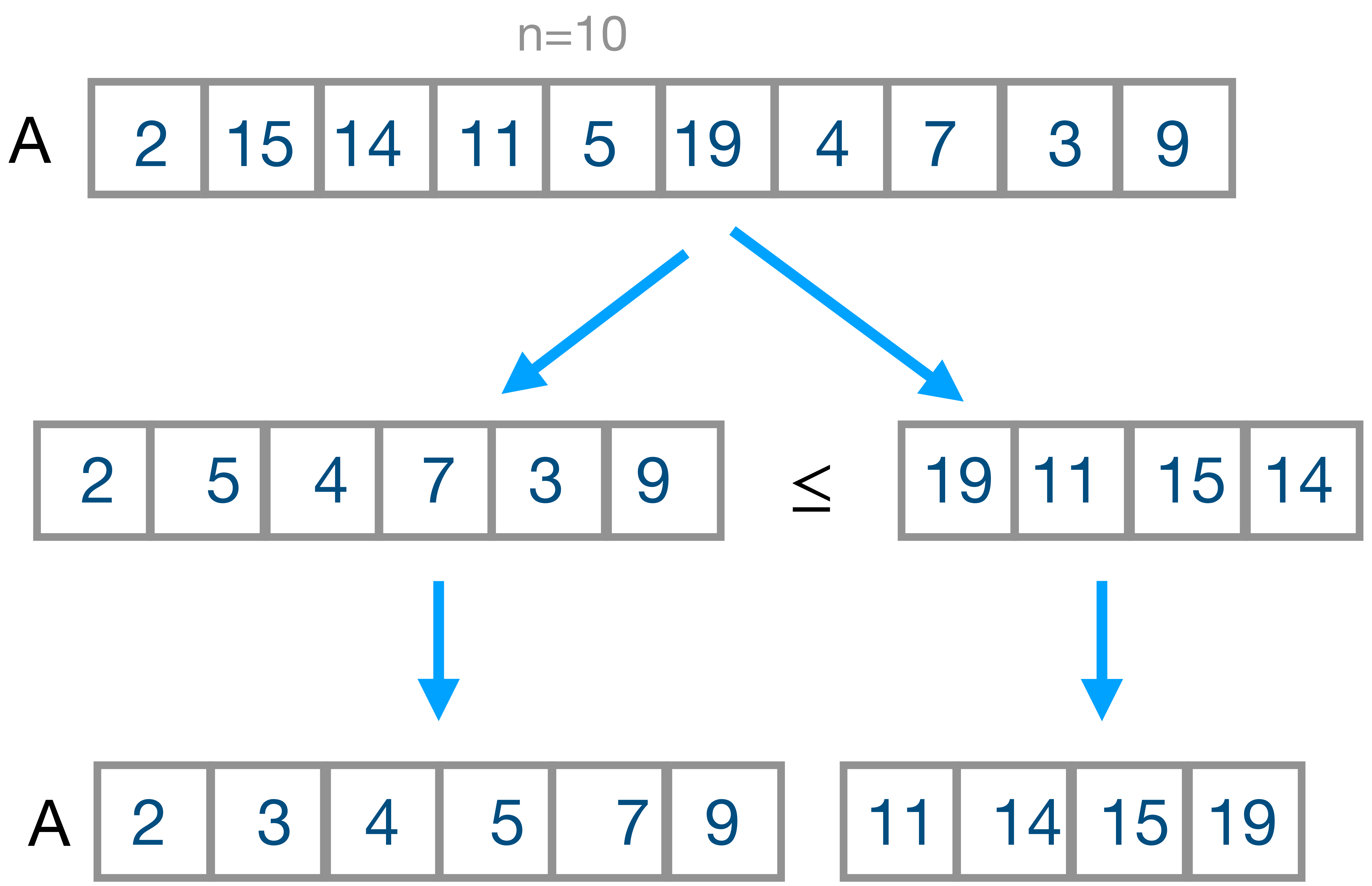
Week 4 Overview

- The priority queue data structure
- The heap
    - Definition, min-heaps and max-heaps
    - Operations: Insert, Delete-Min, Heapify, Buildheap
    - Heapsort
- Quicksort
    - Partition
- Randomized quicksort

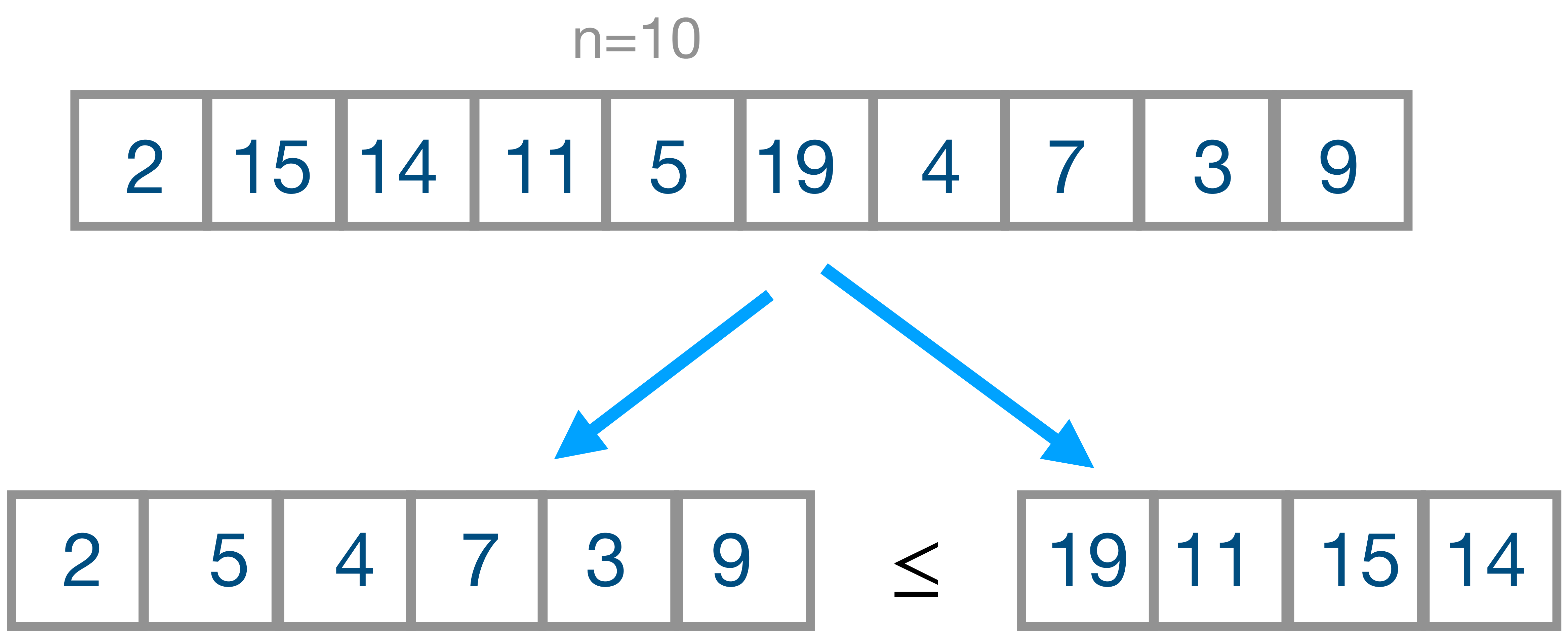
Quicksort

Overview

- **Partition** A: A= [A' | A"], such that A' ≤ A"
- Sort A' recursively
- Sort A" recursively
- Now A = A'A" is sorted   (no need of merging)



How to partition?



Goal: A = A’ | A”, A’ ≤ A”

- Pick an element x of A (called: **the pivot**); put all elements ≤ x in A1, and elements > x in A2
- If we can use extra space, easy (how?)
- How can we partition in place?

Lomuto’s partition

	p									r
A	2	15	14	11	5	19	4	7	3	9

returns an index q, with $p \leq q \leq r$ such that

- $A[i] \leq A[q]$ for all $p \leq i < q$, and
- $A[q] \leq A[i]$ for all $q < i \leq r$

	p					q				r
A	2	5	4	7	3	9	14	11	15	19
			$\leq x$					$> x$		

Lomuto's partition

	p									r
A	2	15	14	11	5	19	4	7	3	9

F

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
FOR j = p TO r - 1 DO
        IF A[j] ≤ x THEN
            i = i + 1
            Exchange A[i] and A[j]
Exchange A[i + 1] and A[r]
RETURN i + 1
```

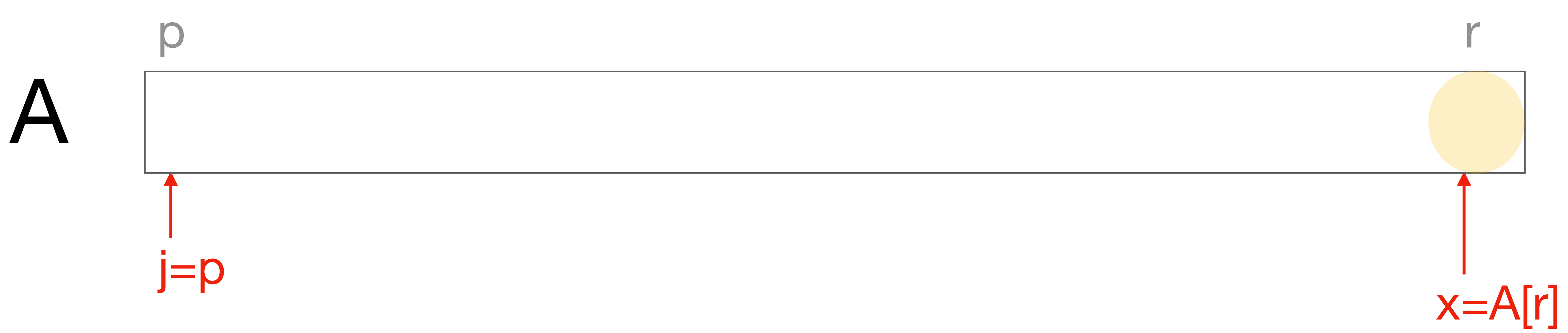
returns an index q, with $p \leq q \leq r$ such that

- $A[i] \leq A[q]$ for all $p \leq i < q$, and
- $A[q] \leq A[i]$ for all $q < i \leq r$

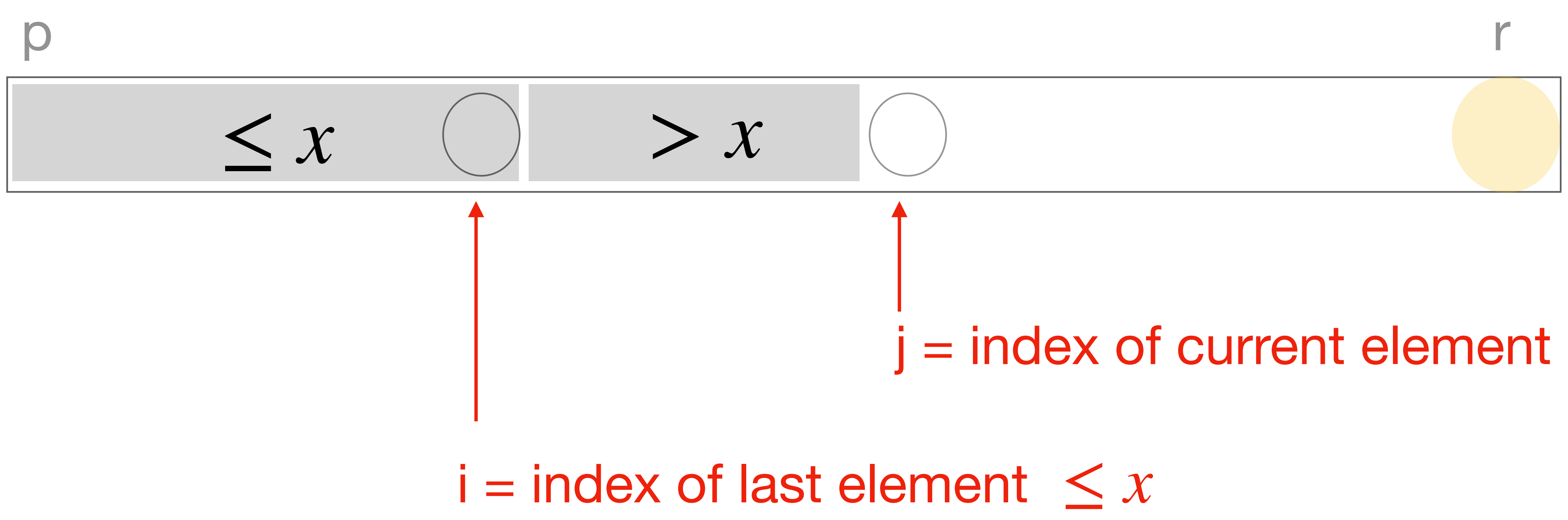
	p					q				r
A	2	5	4	7	3	9	14	11	15	19

$\leq x$ $\qquad$ $> x$

Understanding Lomuto's partition




```
PARTITION(A, p, r)
x = A[r]
i = p − 1
FOR j = p TO r − 1 DO
    IF A[j] ≤ x THEN
        i = i + 1
        Exchange A[i] and A[j]
Exchange A[i + 1] and A[r]
RETURN i + 1
```



- Loop invariants:
  - $A[k] \leq x$ for all $p \leq k \leq i$, and
  - $A[k] > x$ for all $i + 1 \leq k < j$

r

Lomuto's partition

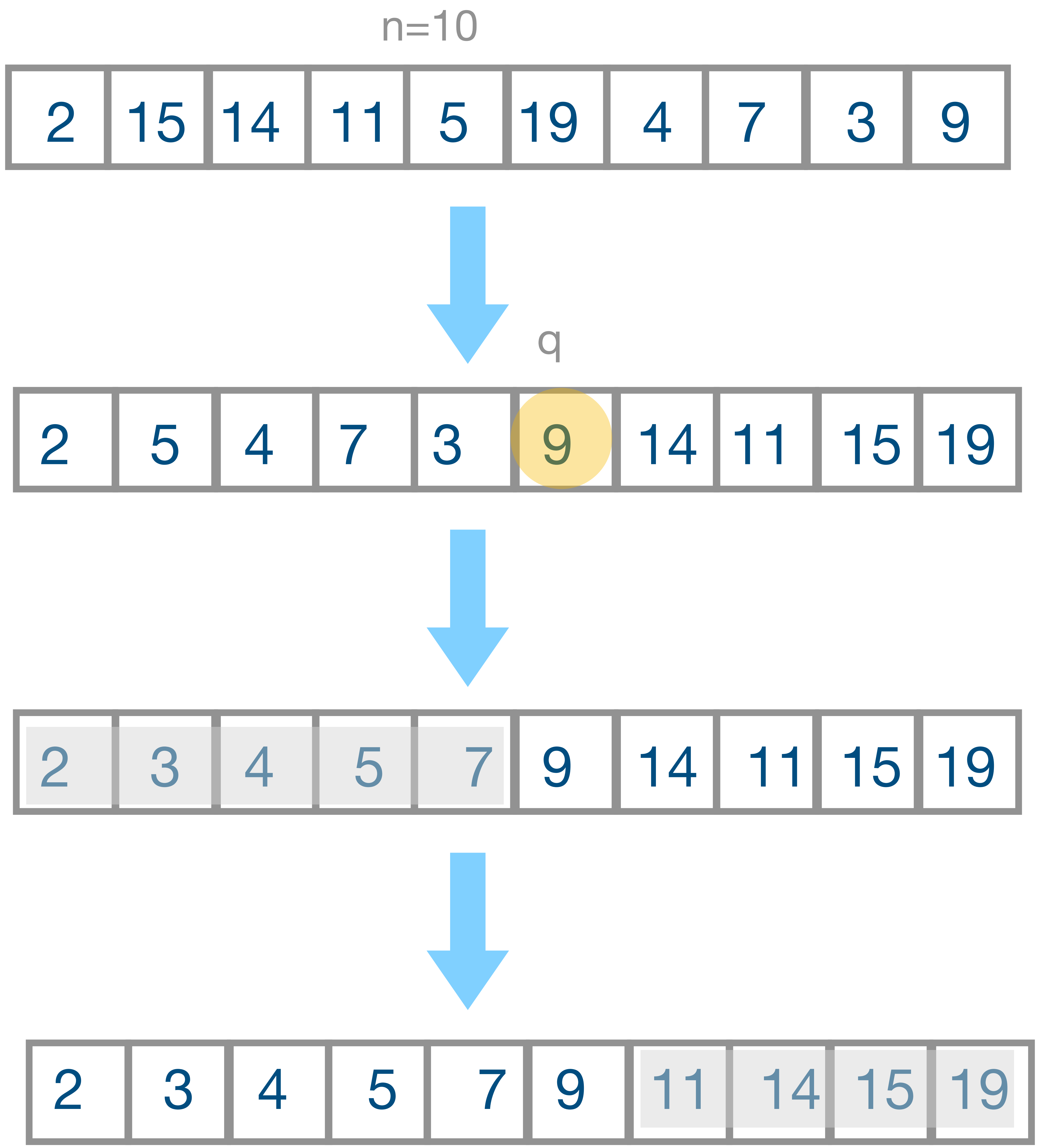
r

$\textsc{Partition}(A, p, r)$
$x = A[r]$
$i = p - 1$
FOR $j = p$ TO $r - 1$ DO
    IF $A[j] \leq x$ THEN
        $i = i + 1$
        Exchange $A[i]$ and $A[j]$
Exchange $A[i + 1]$ and $A[r]$
RETURN $i + 1$

Run time: O(r-p), or O(n) when partitioning the entire array

Quicksort

$\textsc{Quicksort}(A, p, r)$
IF $p < r$ THEN
  q=$\textsc{Partition}(A, p, r)$
  $\textsc{Quicksort}(A, p, q-1)$
  $\textsc{Quicksort}(A, q+1, r)$



Initial call: Quicksort(A, 0, A.size() -1)

Quicksort

Run time: ? need to write a recurrence

$$T(n) = \Theta(n) + T(n_1) + T(n_2)$$

$\textsc{Quicksort}(A, p, r)$
IF $p < r$ THEN

$\quad$ q=$\textsc{Partition}(A, p, r)$

$\quad$ $\textsc{Quicksort}(A, p, q-1)$

$\quad$ $\textsc{Quicksort}(A, q+1, r)$

Quicksort

Run time: ? need to write a recurrence

$$T(n) = \Theta(n) + T(n_1) + T(n_2)$$

$\textsc{Quicksort}(A, p, r)$
IF $p < r$ THEN
  q=$\textsc{Partition}(A, p, r)$
  $\textsc{Quicksort}(A, p, q-1)$
  $\textsc{Quicksort}(A, q+1, r)$

1. Perfectly balanced half-half split for all rec. calls

$$T(n) = \Theta(n) + 2T(n/2) \Longrightarrow \Theta(n \lg n)$$ Best case

Quicksort

Run time: ? need to write a recurrence

$$T(n) = \Theta(n) + T(n_1) + T(n_2)$$

$$
\begin{array}{|l|}
\hline
\textsc{Quicksort}(A, p, r) \\
\text{IF } p < r \text{ THEN} \\
\qquad \text{q=}\textsc{Partition}(A, p, r) \\
\qquad \textsc{Quicksort}(A, p, q-1) \\
\qquad \textsc{Quicksort}(A, q+1, r) \\
\hline
\end{array}
$$

1. Perfectly balanced half-half split for all rec. calls

$$T(n) = \Theta(n) + 2T(n/2) \implies \Theta(n \lg n)$$ Best case

3. All elements on one side of the partition, for all rec calls

$$T(n) = \Theta(n) + T(n-1) \implies \Theta(n^2)$$ Worst case

Average run time ?

- half-half split: $T(n) = \Theta(n) + 2T(n/2) \Longrightarrow \Theta(n \lg n)$
- $\frac{1}{3} - to - \frac{2}{3}$ split: $T(n) = \Theta(n) + T(n/3) + T(2n/3) \Longrightarrow \Theta(n \lg n)$
- $\frac{1}{4} - to - \frac{3}{4}$ split: $T(n) = \Theta(n) + T(n/4) + T(3n/4) \Longrightarrow \Theta(n \lg n)$
- $\frac{1}{10} - to - \frac{9}{10}$ split: $T(n) = \Theta(n) + T(n/10) + T(9n/10) \Longrightarrow \Theta(n \lg n)$
- ….
- MANY good splits !
- Seems that best case will occur often

Average run time

This assumption is important!

Claim: Assuming all input permutations are equally likely, Quicksort average run time is $\Theta(n \lg n)$.

- Intuition:
  - (A lot) more good splits than bad splits
  - Imagine a good split followed by a bad split:
    - good split: n ==> n/2, n/2
    - bad split: n/2 ==> 0, n/2-1
    - Overall, after 2 levels of recursion: n ==> n/2-1, n/2-1
    - It takes **two** recursion levels to half the input
    - Recursion depth will be 2 lg n

Average run time

This assumption is important!

Claim: Assuming all input permutations are equally likely, Quicksort average run time is $\Theta(n \lg n)$.

- Intuition:
  - (A lot) more good splits than bad splits
  - Imagine a good split followed by a bad split:
    - good split: n ==> n/2, n/2
    - bad split: n/2 ==> 0, n/2-1
    - Overall, after 2 levels of recursion: n ==> n/2-1, n/2-1
    - It takes **two** recursion levels to half the input
    - Recursion depth will be 2 lg n

The assumption is not realistic

Randomized Quicksort

$\textsc{RandPartition}(A, p, r)$
i=$\textsc{Random}(p, r)$
Exchange $A[r]$ and $A[i]$
RETURN $\textsc{Partition}(A, p, r)$

$\textsc{RandQuicksort}(A, p, r)$
IF $p < r$ THEN
    q=$\textsc{RandPartition}(A, p, r)$
    $\textsc{RandQuicksort}(A, p, q-1)$
    $\textsc{RandQuicksort}(A, q+1, r)$

Claim: Expected running time of Randomized Quicksort is $\Theta(n \lg n)$ no matter what the input distribution is.

Quicksort: summary

- Quicksort:
  - Best case $\Theta(n)$
  - Worst-case $\Theta(n^2)$
  - Average case $\Theta(n \lg n)$ if all input permutations are equally likely
- Randomized Quicksort:
  - Best case $\Theta(n)$
  - Worst-case $\Theta(n^2)$
  - Expected case $\Theta(n \lg n)$ on any sets of inputs
- In place
- Randomized-Quicksort is the fastest sort in practice (on general inputs)