CS473-Algorithms I

Lecture 5

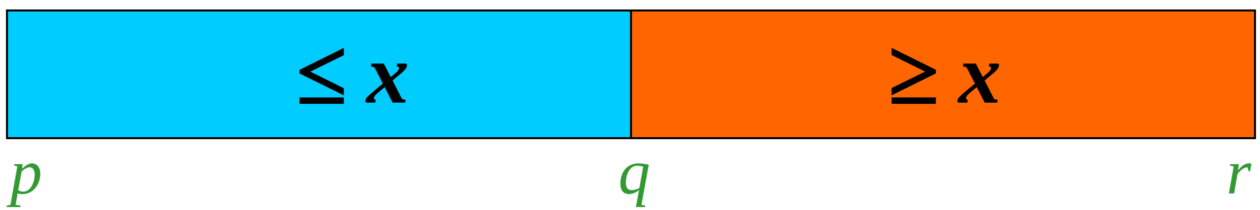
Quicksort

Quicksort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts "in place" (like insertion sort, but not like merge sort).
- Very practical (with tuning).

Quicksort

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

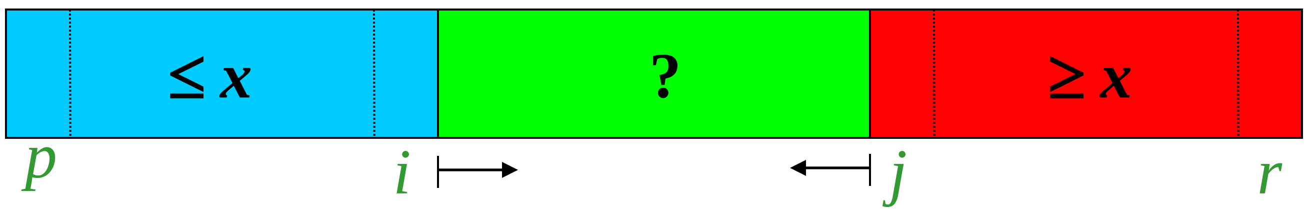


2. Conquer: Recursively sort 2 subarrays
3. Combine: Trivial (because in-place)

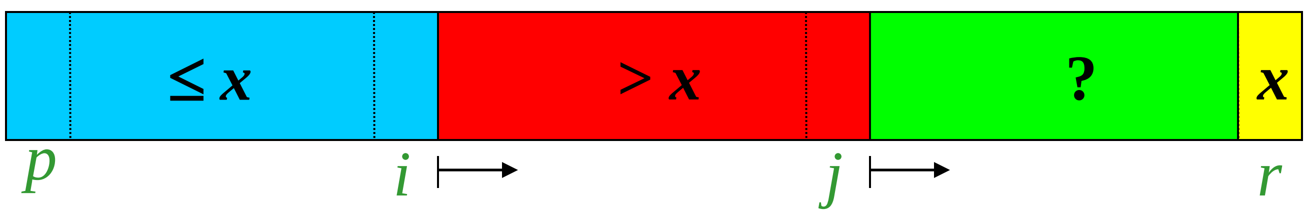
- Key: Linear-time ($\Theta(n)$) partitioning algorithm

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 ($pivot = x = A[r]$)



Hoare's Partitioning Algorithm

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 $A[i] \geq pivot$

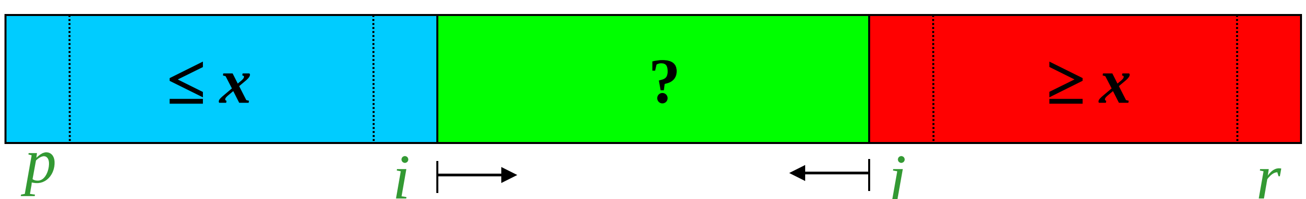
if $i < j$ then

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

else

return j

Running time is $O(n)$



QUICKSORT (A, p, r)

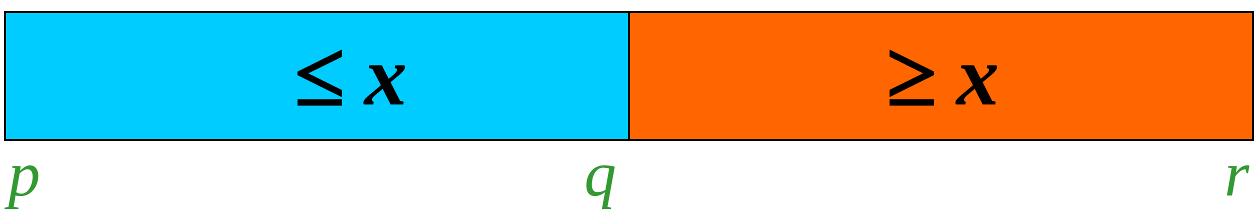
  if $p < r$ then

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

    QUICKSORT(A, p, q)

    QUICKSORT(A, q +1, r)

Initial invocation: QUICKSORT(A, 1, n)



Hoare's Partitioning Algorithm

- Select a pivot element: *pivot*=A[*p*] from A[*p*…*r*]
- Grows two regions

    A[*p*…*i*] from left to right

    A[*j*…*r*] from right to left

  such that

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

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

Hoare's Partitioning Algorithm

- The two regions A[$p \ldots i$] and A[$j \ldots r$] grow until

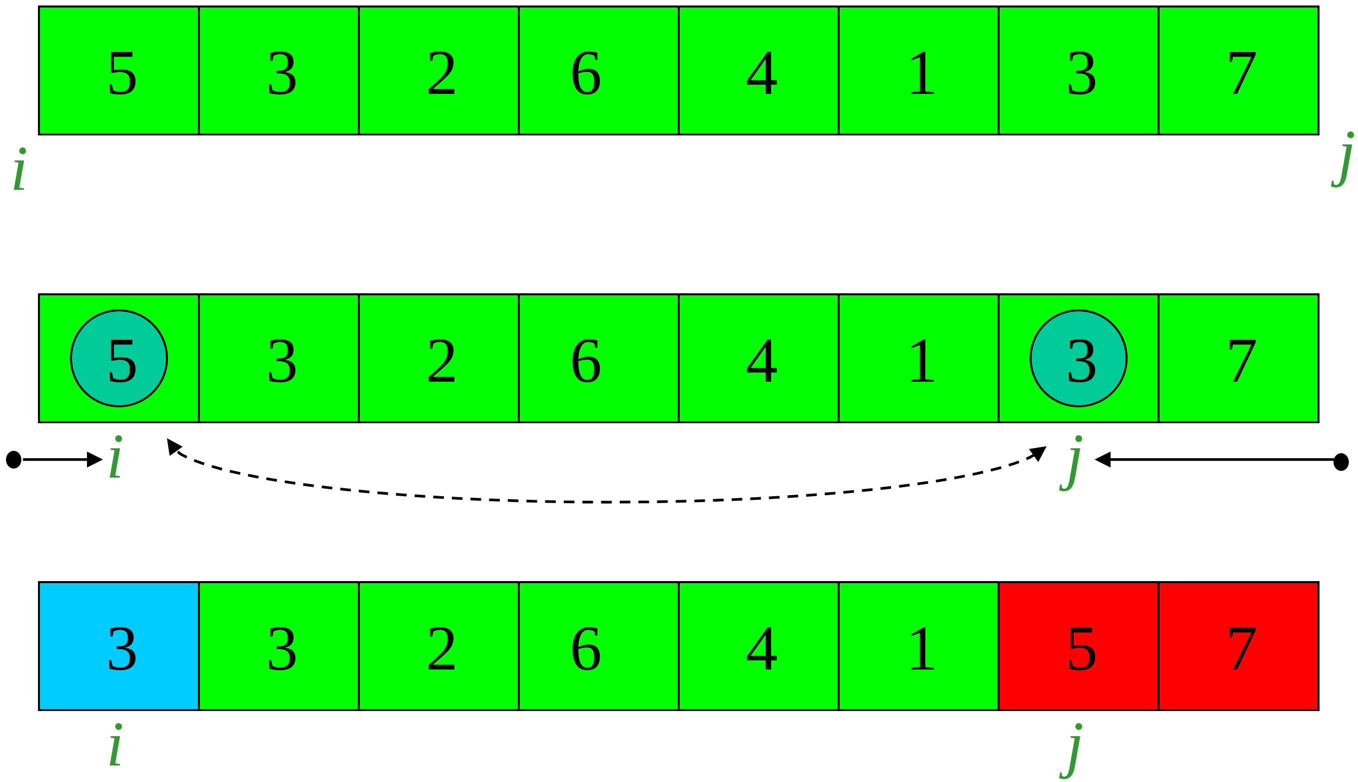
  $A[i] \geq pivot \geq A[j]$

- Assuming these inequalities are strict
  - A[i] is too large to belong to the left region
  - A[j] is too small to belong to the right region
  - exchange $A[i] \leftrightarrow A[j]$ for future growing in the next iteration

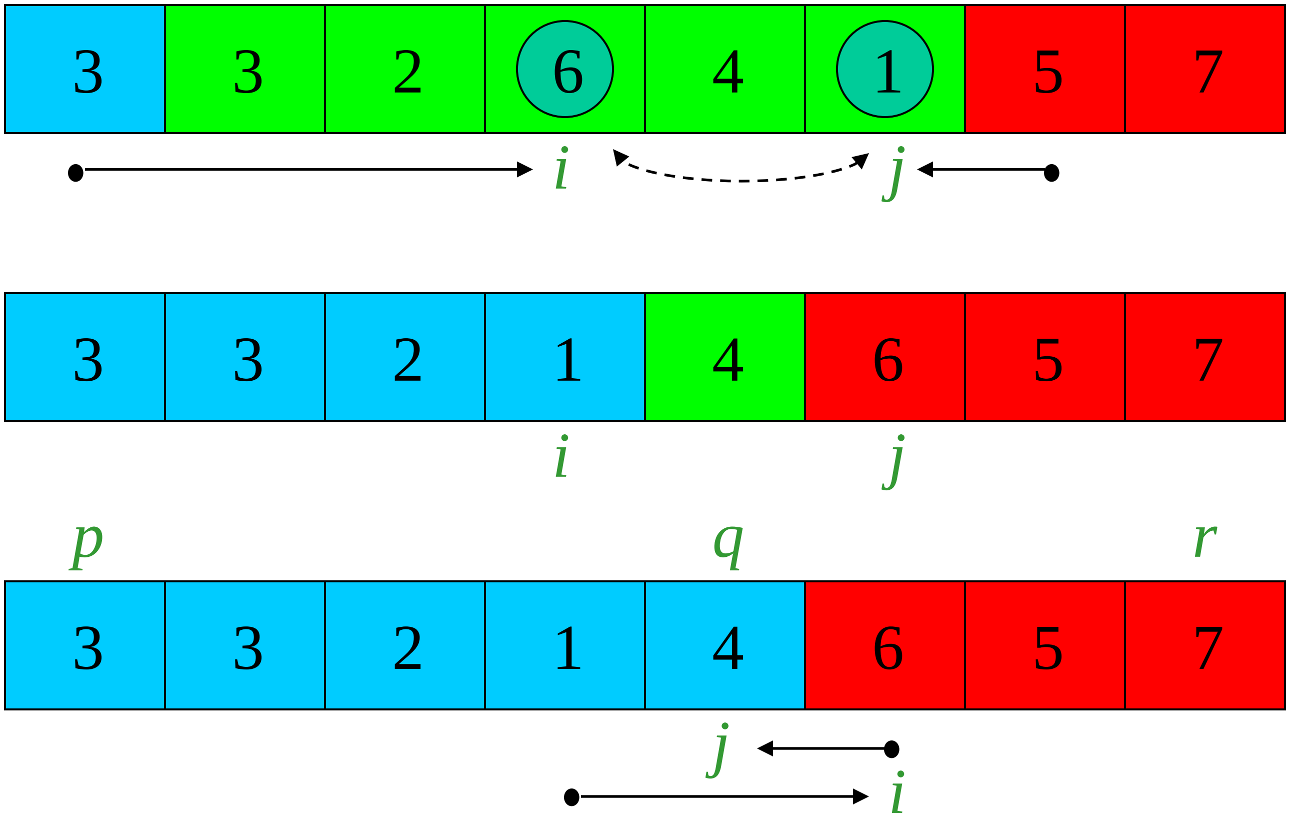
Hoare's Partitioning Algorithm

- It is important that
  - A[p] is chosen as the pivot element
  - If A[r] is used as pivot then
    - may yield a trivial split (termination $i = j = r$)
    - occurs when A[$p \ldots r-1$] $<$ *pivot* $=$ A[r]
    - then quicksort may loop forever since $q = r$

Hoare's Algorithm: Example 1 (pivot = 5)

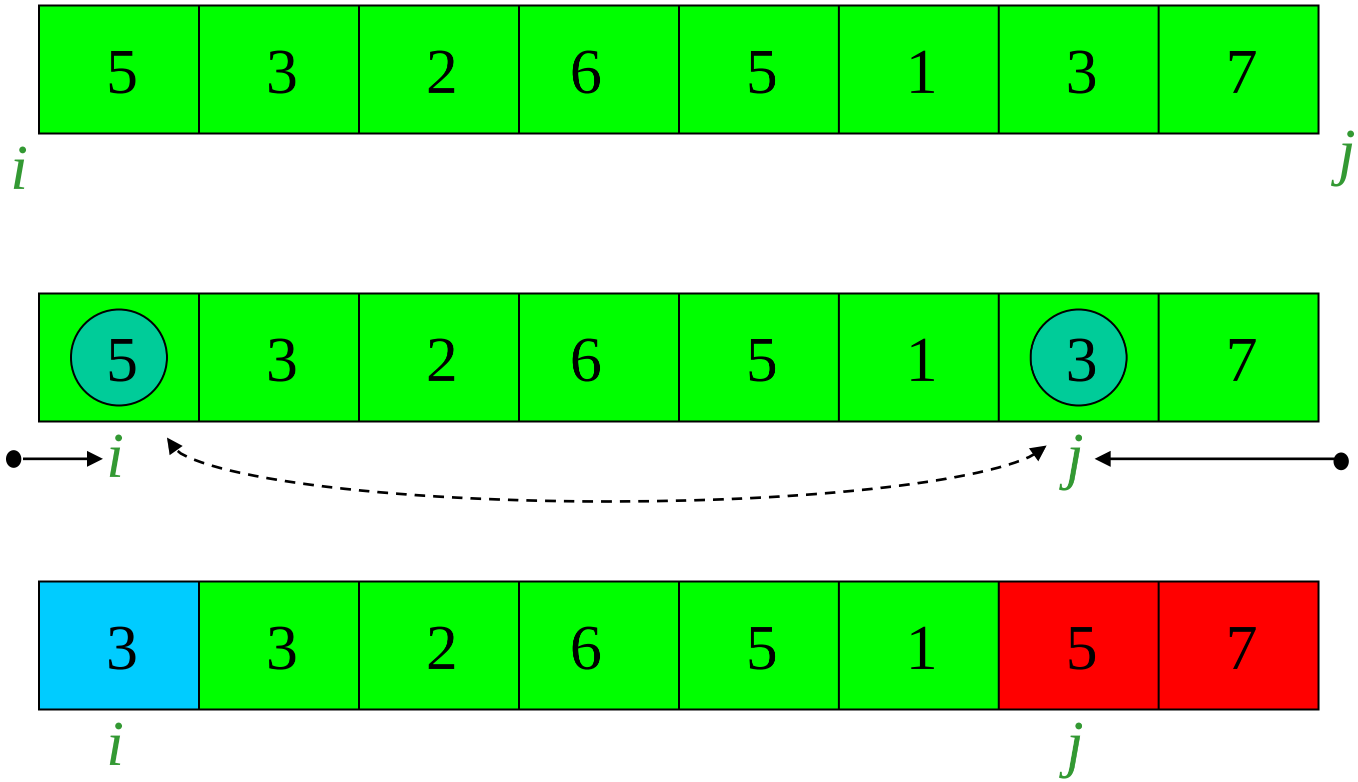


Hoare's Algorithm: Example 1 (pivot = 5)

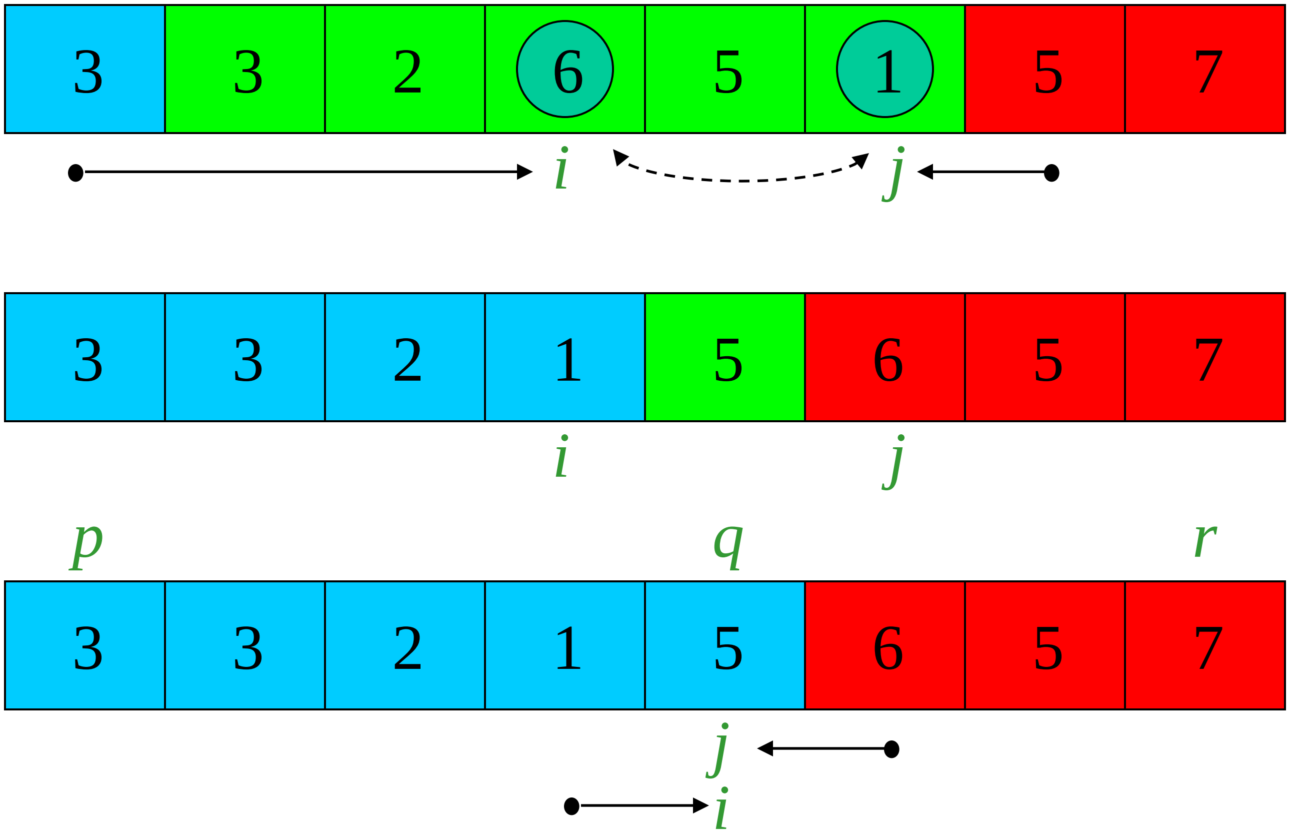


Termination: $i = 6; j = 5,\quad$ i.e., $i = j + 1$

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



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



Termination: $i = j = 5$

Correctness of Hoare's Algorithm

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

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

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

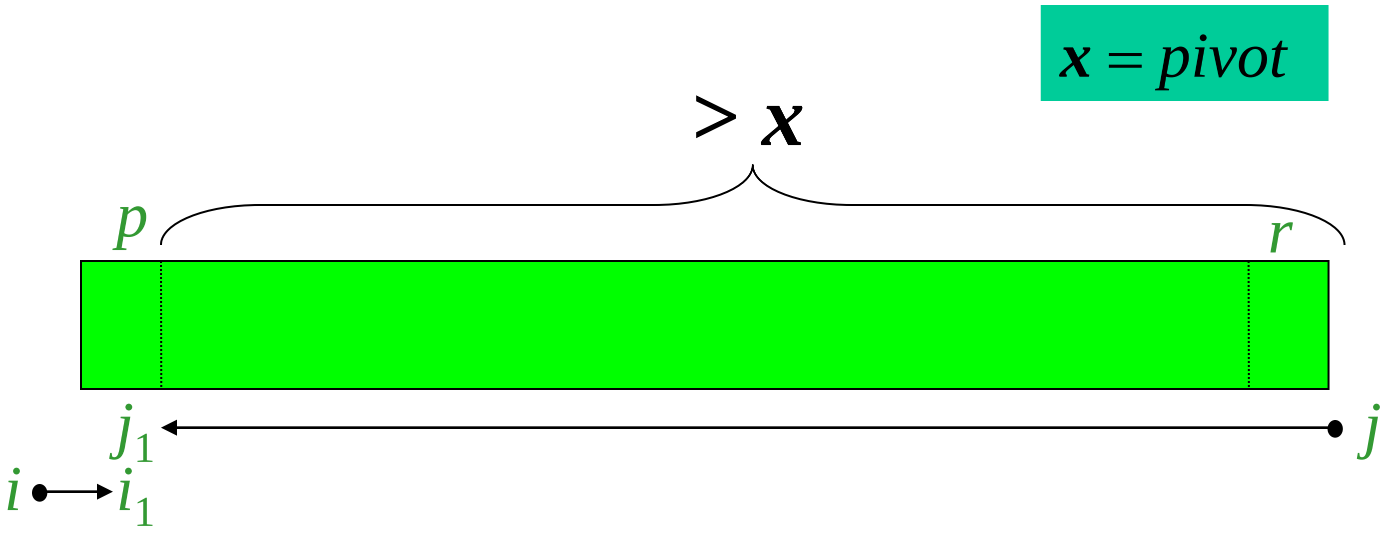
Notation used for proof:

- k = # of times while-loop iterates until termination
- i_l & j_l = values of i & j indices at the end of iteration $1 \leq l \leq k$
- Note: we always have $i_1 = p$ & $p \leq j_1 \leq r$

Correctness of Hoare's Algorithm

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

- $k = 1$: occurs when A[p+1…r] > *pivot* $\Rightarrow$ $i_1 = j_1 = p$

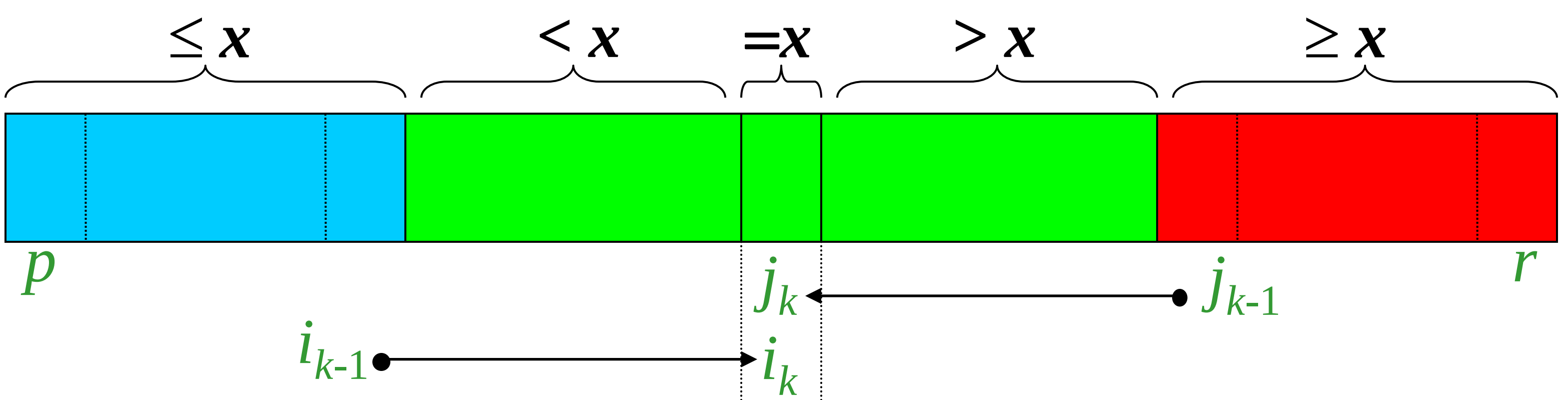


Correctness of Hoare's Algorithm

- $k > 1$: we have $i_{k\text{-}1} < j_{k\text{-}1}$
  ($A[i_{k\text{-}1}] \leq pivot$ & $A[j_{k\text{-}1}] \geq pivot$ due to exchange)

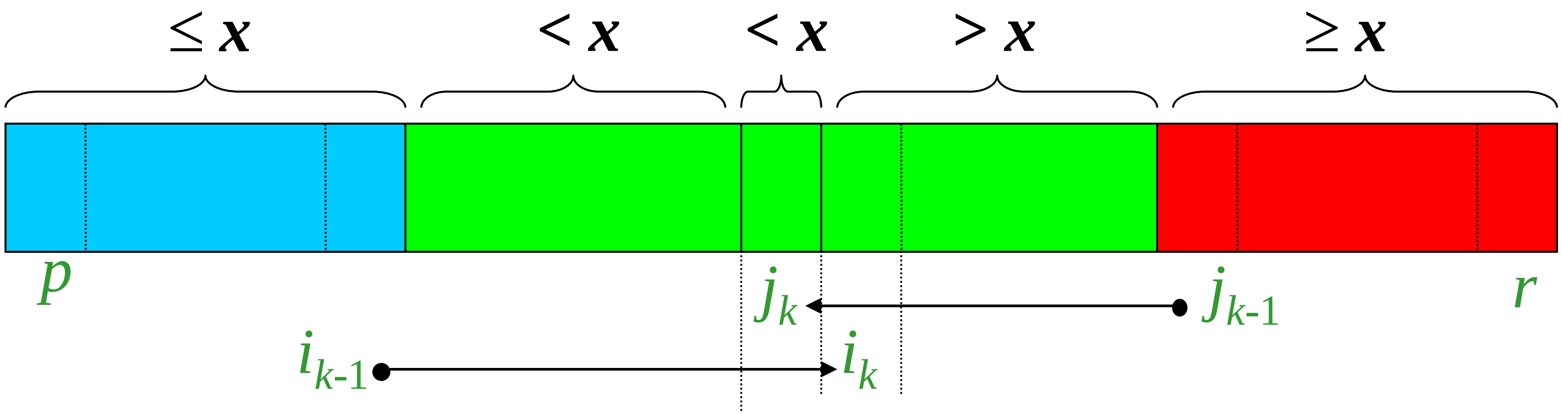
$x = pivot$

- case a: $i_{k\text{-}1} < j_k < j_{k\text{-}1}$
  - case a-1: $A[j_k] = pivot \Rightarrow i_k = j_k$



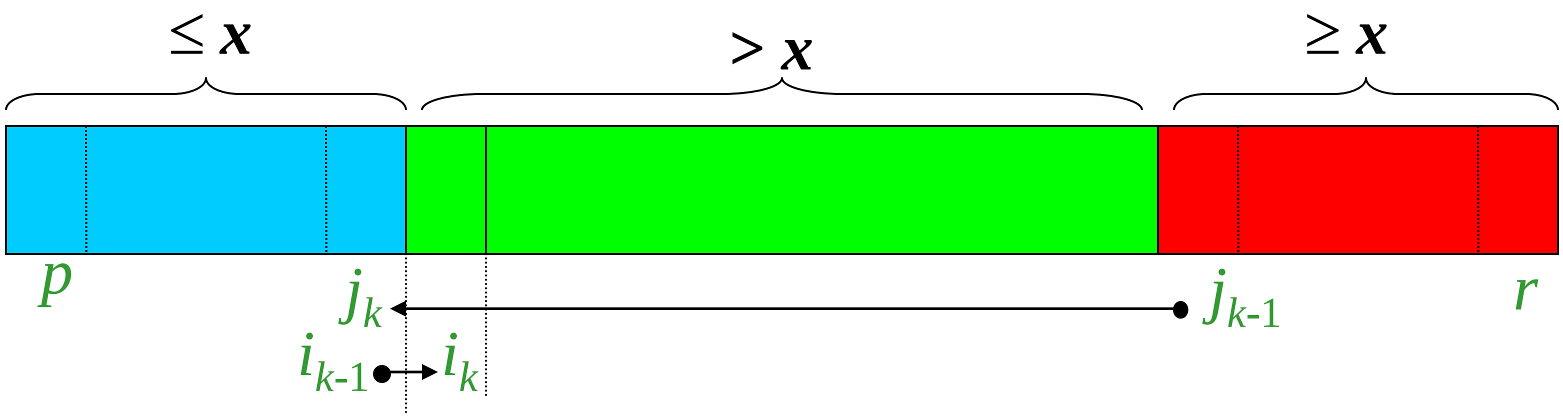
– case a-2: $A[j_k] < pivot \Rightarrow i_k = j_k + 1$

$x = pivot$



• case b: $i_{k-1} = j_k < j_{k-1} \Rightarrow i_k = j_k + 1$

$x = pivot$



Correctness of Hoare's Algorithm

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

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

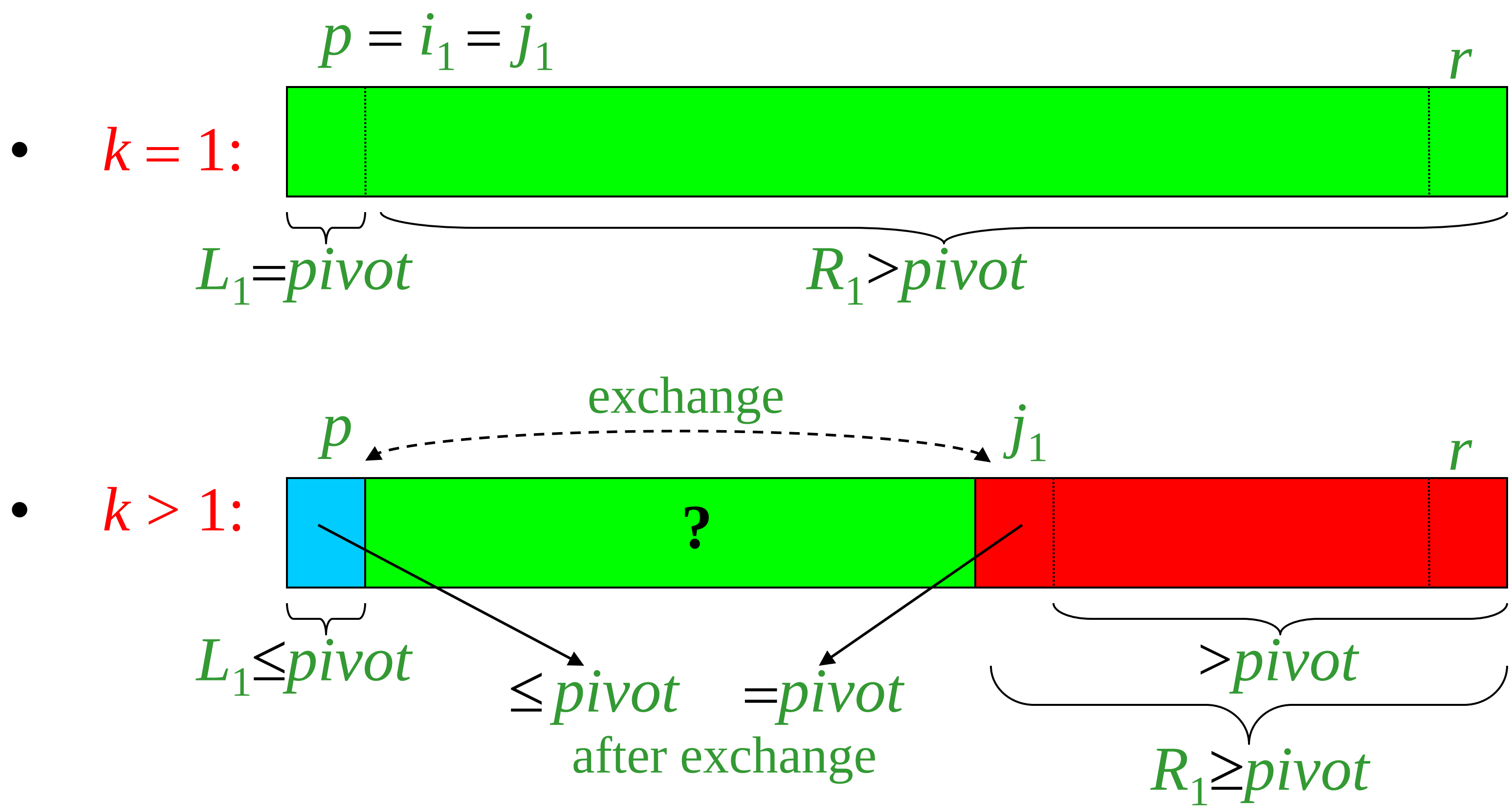
Proof of (a) & (b)

- $k = 1$: trivial since $i_1 = j_1 = p$
- $k > 1$: $p \leq j_k < r \Rightarrow p < i_k \leq r$ due to the lemma

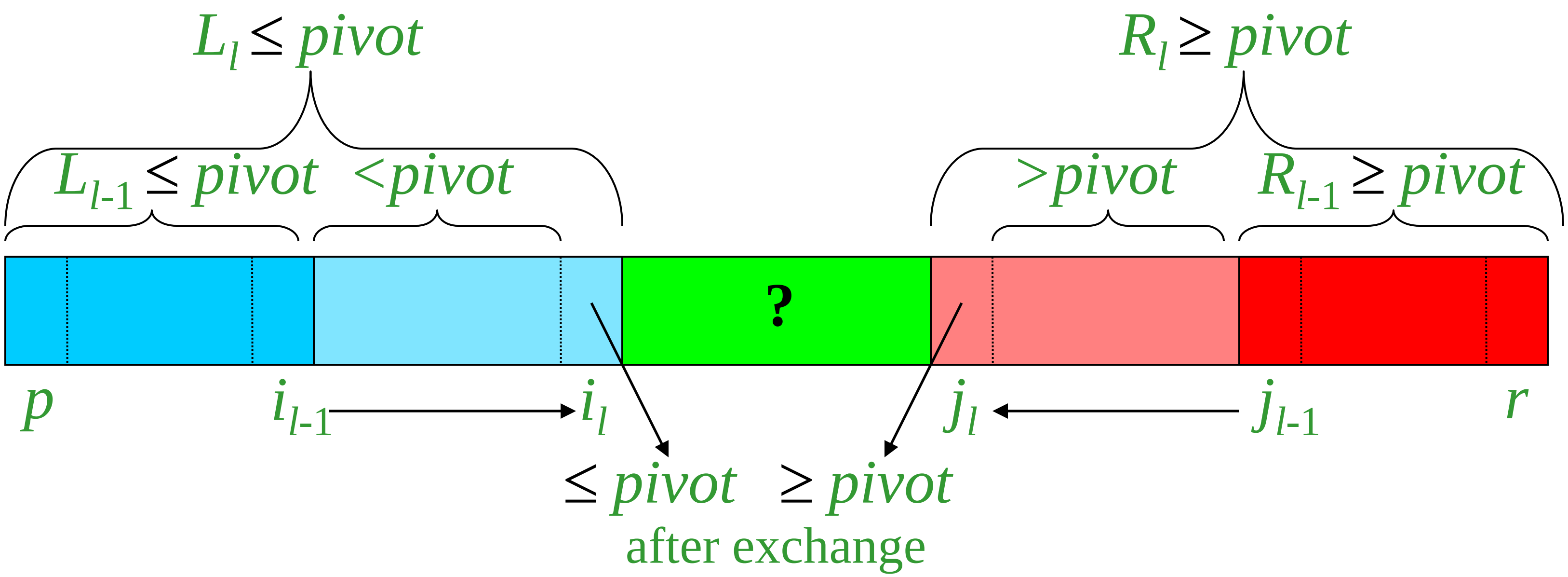
(c) Every element in A[$p \ldots j$] $\leq$ every element in A[j +1…r] at termination

Proof of (c) : by induction on l (while-loop iteration sequence)

Basis: true for $l = 1$



- Hypothesis: $L_{l\text{-}1} = \mathrm{A}[p\ldots\ i_{l\text{-}1}] \leq pivot \leq \mathrm{A}[j_{l\text{-}1}\ldots r] = R_{l\text{-}1}$
- Show that
  - $L_l = \mathrm{A}[p\ldots\ i_l] \leq pivot \leq \mathrm{A}[j_l\ldots r] = R_l$ if $l < k$
  - $L_l = \mathrm{A}[p\ldots\ j_l] \leq pivot \leq \mathrm{A}[j_l+1\ldots r] = R_l$ if $l = k$
- Case: $l\ < k$ (partition does not terminate at iteration l)



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

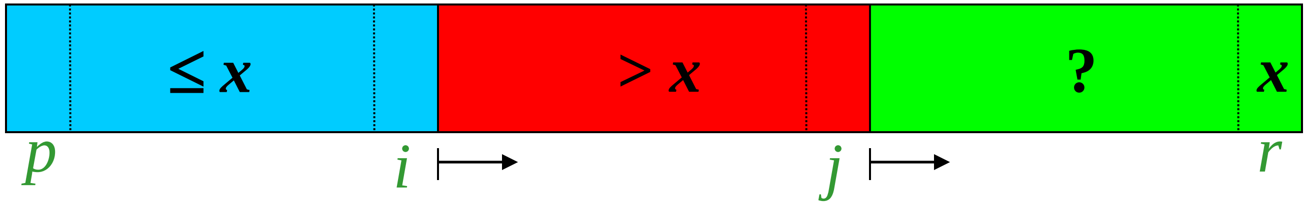
    $i \leftarrow i + 1$

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

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

return $i + 1$

Running time is $O(n)$



QUICKSORT (A, p, r)

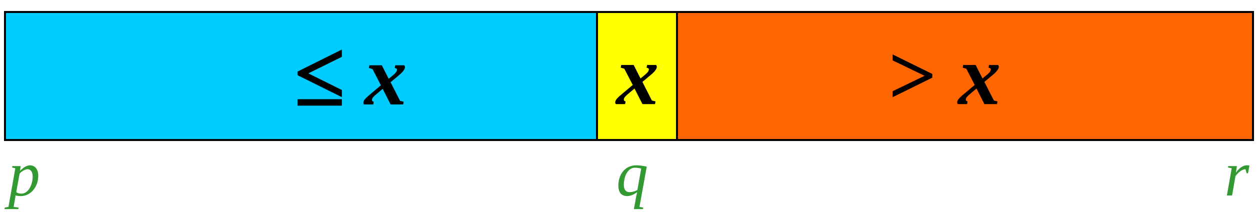
    if $p < r$ then

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

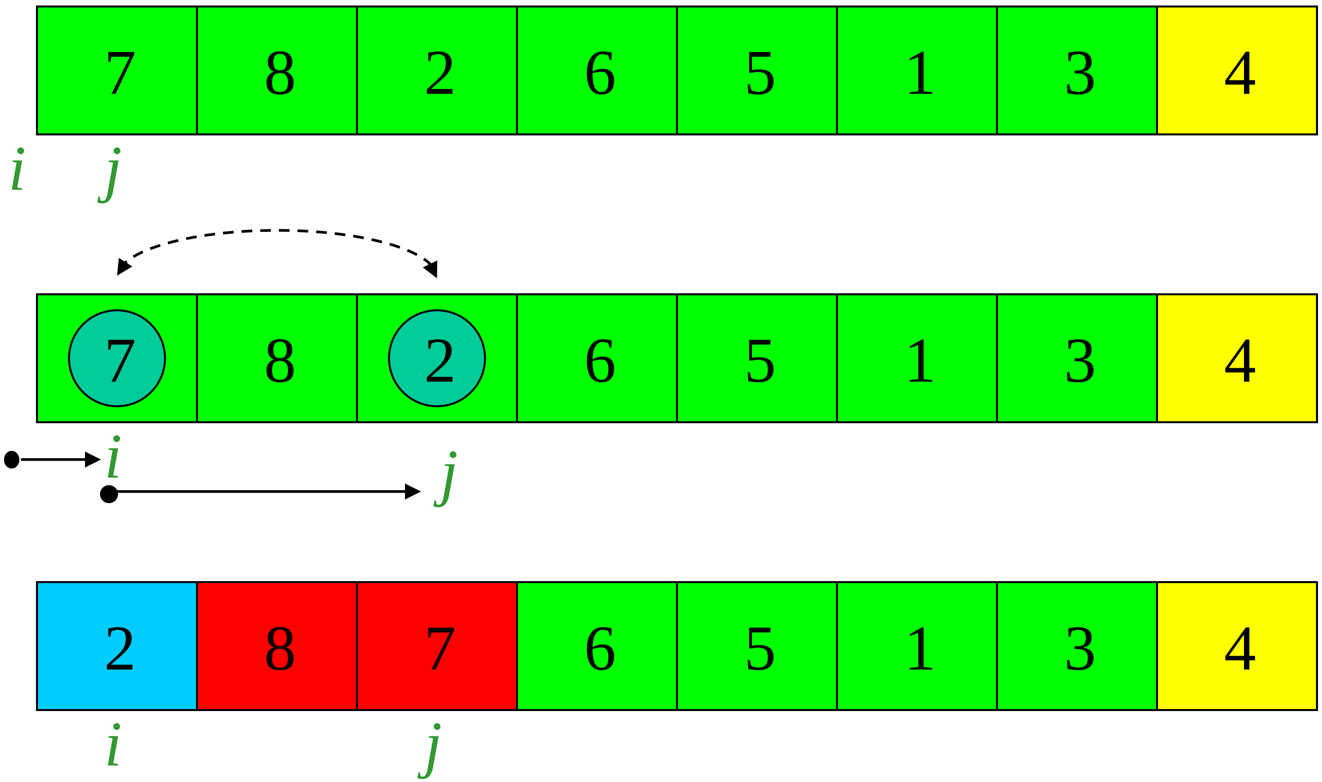
        QUICKSORT(A, p, $q - 1$)

        QUICKSORT(A, $q + 1$, r)

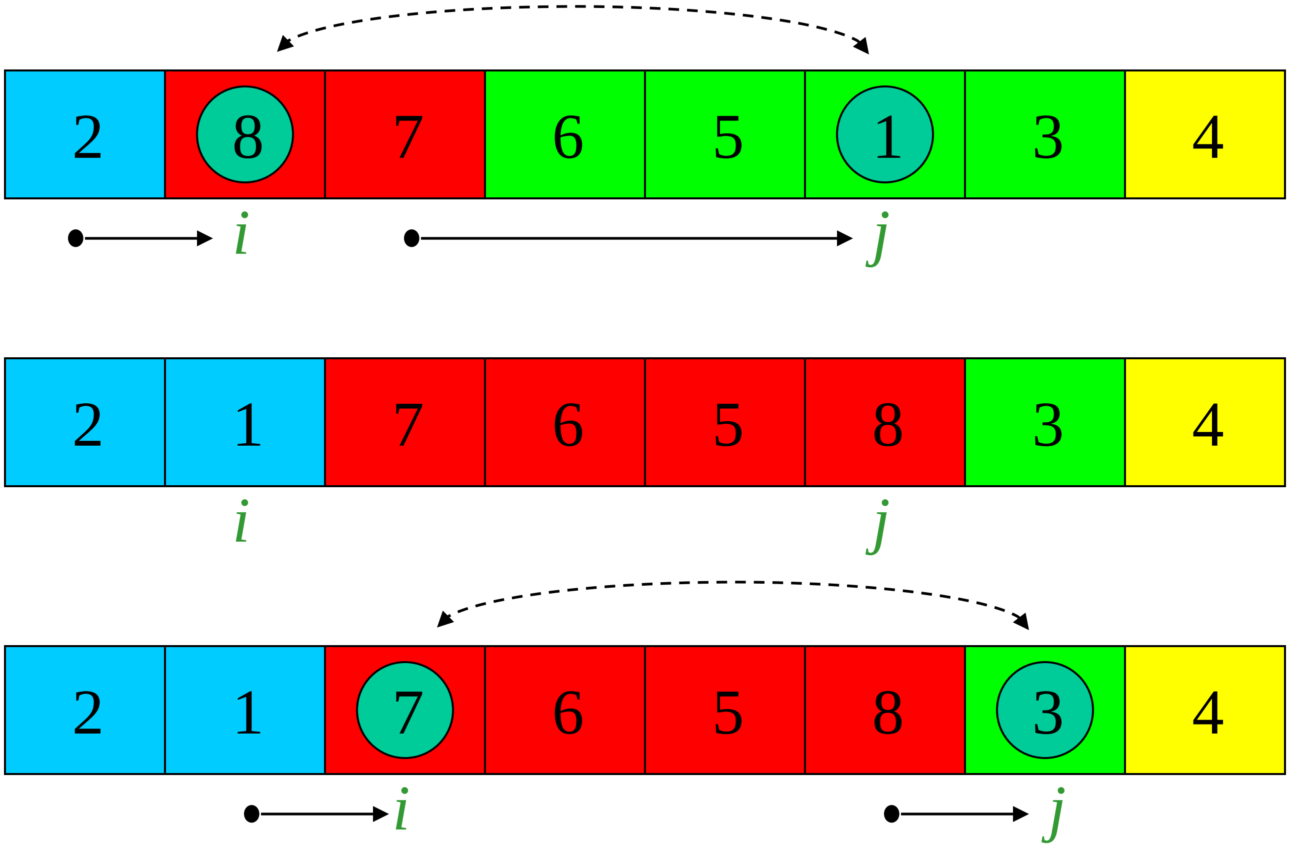
Initial invocation: QUICKSORT(A, 1, n)



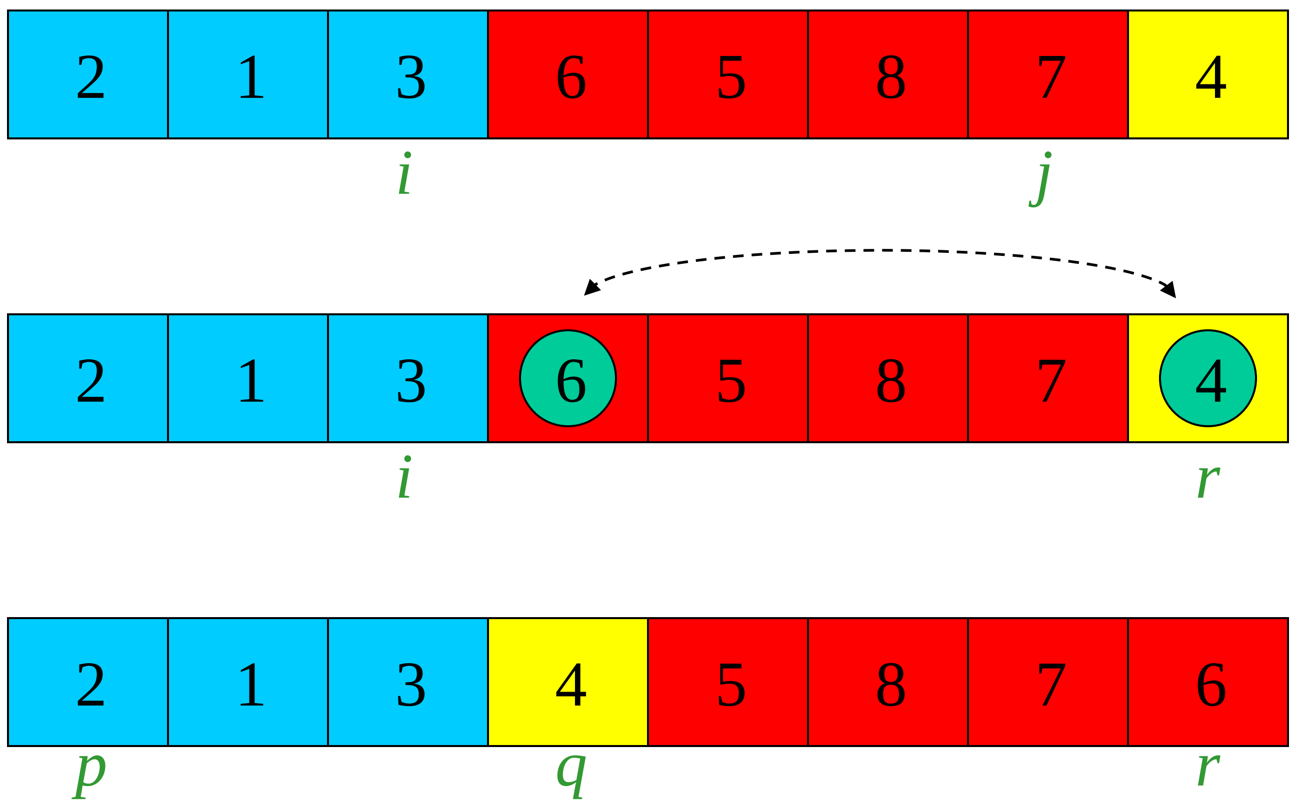
Lomuto's Algorithm: Example (pivot = 4)



Lomuto's Algorithm: Example (pivot = 4)



Example: pivot = 4



Comparison of Hoare's & Lomuto's Algorithms

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

& $pivot = \mathrm{A}[r]$ (Lomuto)

- # of element exchanges: $e(n)$
  - Hoare: $0 \leq e(n) \leq \left\lfloor \frac{n}{2} \right\rfloor$
    - Best: $k = 1$ with $i_1 = j_1 = p$ (i.e., $\mathrm{A}[\,p+1 \ldots r\,] > pivot$)
    - Worst: $\mathrm{A}[\,p+1 \ldots p + \left\lfloor \frac{n}{2} \right\rfloor - 1] \geq pivot \geq \mathrm{A}[\,p + \left\lceil \frac{n}{2} \right\rceil \ldots r\,]$
  - Lomuto: $1 \leq e(n) \leq n$
    - Best: $\mathrm{A}[\,p \ldots r-1\,] > pivot$
    - Worst: $\mathrm{A}[\,p \ldots r-1\,] \leq pivot$

Comparison of Hoare's & Lomuto's Algorithms

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

- # of index comparisons: $c_i(n)$
  - Hoare: $1 \leq c_i(n) \leq \left\lfloor \frac{n}{2} \right\rfloor + 1 \quad (c_i(n) = e(n) + 1)$
  - Lomuto: $c_i(n) = n - 1$

Comparison of Hoare's & Lomuto's Algorithms

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

- 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