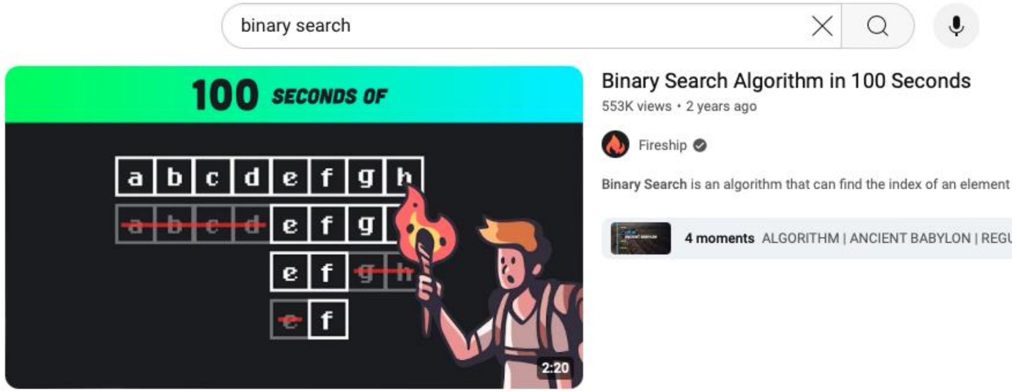
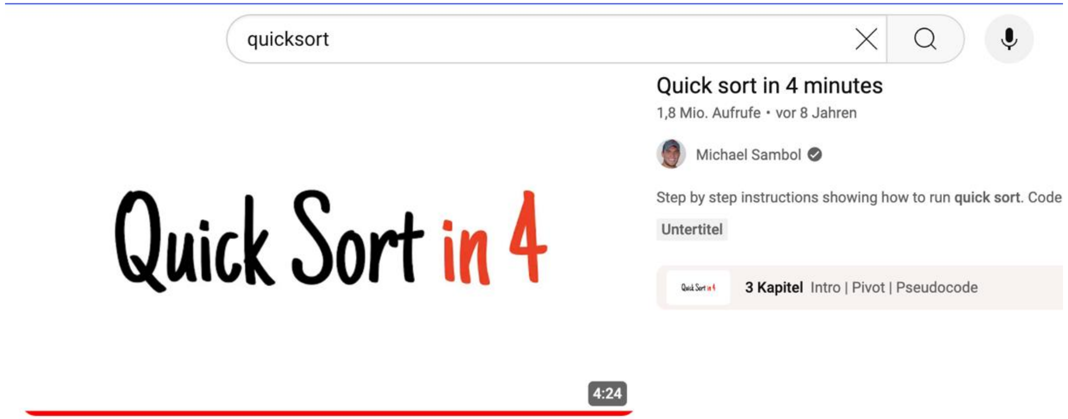
Hello Class!

If you aren't familiar with **binary search** & **quick sort** or would like a quick refresher, please watch the following videos:

1) https://www.youtube.com/watch?v=MFhxShGxHWc (you can start from 1:00)
2) https://www.youtube.com/watch?v=Hoixgm4-P4M





Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

$$
\begin{array}{l}
lo, hi \leftarrow 0, n \\
\textbf{while } lo < hi \\
\quad m \leftarrow (lo + hi)/2 \\
\quad \textbf{if } x < A[m] \\
\quad\quad hi \leftarrow m \\
\quad \textbf{else if } x > A[m] \\
\quad\quad lo \leftarrow m + 1 \\
\quad \textbf{else} \\
\quad\quad \textbf{return } m \\
\textbf{return } not\_found
\end{array}
$$

Slide 24 of lec05

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
-3	0	4	6	8	9	11

lo = 0 m = 3
hi = 7

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x < A[m]$
    $hi \leftarrow m$
  else if $x > A[m]$
    $lo \leftarrow m + 1$
  else
    return m
return not_found

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
-3	0	4	6	8	9	11
			A[m]			

lo = 0 m = 3
hi = 7

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x < A[m]$
    $hi \leftarrow m$
  else if $x > A[m]$
    $lo \leftarrow m + 1$
  else
    return m
return *not_found*

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
-3	0	4	6	8	9	11
			A[m]			

lo = 0 m = 3
hi = 7

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x < A[m]$
    $hi \leftarrow m$
  else if $x > A[m]$
    $lo \leftarrow m + 1$
  else
    return m
return not_found

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
-3	0	4	6	8	9	11
			A[m]			

lo = 0 m = 3

hi = 7

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x < A[m]$
    $hi \leftarrow m$
  else if $x > A[m]$
    $lo \leftarrow m + 1$
  else
    return m
return not_found

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]

[-3 0 4 6 8 9 11] lo = 0 m = 3
A[m] hi = 7

[-3 0 4 6 8 9 11] lo = 4
hi = 7

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x < A[m]$
    $hi \leftarrow m$
  else if $x > A[m]$
    $lo \leftarrow m + 1$
  else
    return m
return not_found

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]			
[	-3	0	4	6	8	9	11	]	lo = 0	m = 3
				A[m]					hi = 7	
[	-3	0	4	6	8	9	11	]	lo = 4	m = 5
						A[m]			hi = 7	

$lo, hi \leftarrow 0, n$
while $lo < hi$
$\quad m \leftarrow (lo + hi)/2$
$\quad$**if** $x < A[m]$
$\quad\quad hi \leftarrow m$
$\quad$**else if** $x > A[m]$
$\quad\quad lo \leftarrow m + 1$
$\quad$**else**
$\quad\quad$**return** m
return not_found

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]		
[-3	0	4	6	8	9	11]	lo = 0	m = 3
			A[m]				hi = 7	
[-3	0	4	6	8	9	11]	lo = 4	m = 5
					A[m]		hi = 7	

$lo, hi \leftarrow 0, n$
while $lo < hi$
    $m \leftarrow (lo + hi)/2$
    if $x < A[m]$
        $hi \leftarrow m$
    else if $x > A[m]$
        $lo \leftarrow m + 1$
    else
        return m
return not_found

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]			
[	-3	0	4	6	8	9	11	]	lo = 0	m = 3
				A[m]					hi = 7	
[	-3	0	4	6	8	9	11	]	lo = 4	m = 5
						A[m]			hi = 7	
[	-3	0	4	6	8	9	11	]	lo = 4	
									hi = 5	

```
lo, hi ← 0, n
while lo < hi
    m ← (lo + hi)/2
    if x < A[m]
        hi ← m
    else if x > A[m]
        lo ← m + 1
    else
        return m
return not_found
```

Slide 24 of lec05

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]			
[	-3	0	4	6	8	9	11	]	lo = 0	m = 3
				A[m]					hi = 7	
[	-3	0	4	6	8	9	11	]	lo = 4	m = 5
						A[m]			hi = 7	
[	-3	0	4	6	8	9	11	]	lo = 4	m = 4
					A[m]				hi = 5	

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x < A[m]$
    $hi \leftarrow m$
  else if $x > A[m]$
    $lo \leftarrow m + 1$
  else
    return m
return not_found

Slide 24 of lec05

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]			
[-3	0	4	6	8	9	11]	lo = 0	m = 3
			A[m]				hi = 7	
[-3	0	4	6	8	9	11]	lo = 4	m = 5
					A[m]		hi = 7	
[-3	0	4	6	8	9	11]	lo = 4	m = 4
				A[m]			hi = 5	

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x < A[m]$
    $hi \leftarrow m$
  else if $x > A[m]$
    $lo \leftarrow m + 1$
  else
    return m
return *not_found*

Slide 24 of lec05

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]			
[	-3	0	4	6	8	9	11	]	lo = 0	m = 3
				A[m]					hi = 7	
[	-3	0	4	6	8	9	11	]	lo = 4	m = 5
						A[m]			hi = 7	
[	-3	0	4	6	8	9	11	]	lo = 4	m = 4
					A[m]				hi = 5	

$lo, hi \leftarrow 0, n$
while $lo < hi$
    $m \leftarrow (lo + hi)/2$
    if $x < A[m]$
        $hi \leftarrow m$
    else if $x > A[m]$
        $lo \leftarrow m + 1$
    else
        return m
return *not_found*

Slide 24 of lec05

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

search for 8

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]		
-3	0	4	6	8	9	11	lo = 0, hi = 7	m = 3
			A[m]					
-3	0	4	6	8	9	11	lo = 4, hi = 7	m = 5
					A[m]			
-3	0	4	6	8	9	11	lo = 4, hi = 5	m = 4
				A[m]				

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x < A[m]$
    $hi \leftarrow m$
  else if $x > A[m]$
    $lo \leftarrow m + 1$
  else
    return m
return not_found

Slide 24 of lec05

Binary Search

- when searching a **sorted** array for an object
- partition array at the center
- either discard the right half or the left half repeatedly

- Asymptotic cost: $O(\log n)$

search for 8

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]			
[	-3	0	4	6	8	9	11	]	lo = 0	m = 3
				A[m]					hi = 7	
[	-3	0	4	6	8	9	11	]	lo = 4	m = 5
						A[m]			hi = 7	
[	-3	0	4	6	8	9	11	]	lo = 4	m = 4
					A[m]				hi = 5	

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x < A[m]$
    $hi \leftarrow m$
  else if $x > A[m]$
    $lo \leftarrow m + 1$
  else
    return m
return not_found

Slide 24 of lec05

Exercise 1 lec05

Exercise 1

Suppose that the sorted array may contain duplicate items. Linear search will find the *first* match.

Modify the binary search algorithm so that it also identifies the first match if there are duplicates.

Modify the demonstration of correctness so that it matches your altered algorithm.

Exercise 1 lec05

search for 8

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
-3	0	4	6	8	8	11
			A[m]			

lo = 0 m = 3
hi = 7

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x < A[m]$
    $hi \leftarrow m$
  else if $x > A[m]$
    $lo \leftarrow m + 1$
  else
    return m
return not_found

Exercise 1 lec05

search for 8

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]			
[	-3	0	4	6 A[m]	8	8	11	]	lo = 0 hi = 7	m = 3
[	-3	0	4	6	8	8 A[m]	11	]	lo = 4 hi = 7	m = 5

- How can we modify the code so to identify first match if there are duplicates?

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x < A[m]$
    $hi \leftarrow m$
  else if $x > A[m]$
    $lo \leftarrow m + 1$
  else
    return m
return not_found

Slide 24 of lec05

Exercise 1 lec05

- How can we modify the code so to identify first match if there are duplicates?

$$
\begin{array}{l}
lo, hi \leftarrow 0, n \\
\textbf{while } lo < hi \\
\quad m \leftarrow (lo + hi)/2 \\
\quad \textbf{if } x < A[m] \\
\quad\quad hi \leftarrow m \\
\quad \textbf{else if } x > A[m] \\
\quad\quad lo \leftarrow m + 1 \\
\quad \textbf{else} \\
\quad\quad \textbf{return } m \\
\textbf{return } not\_found
\end{array}
$$

Slide 24 of lec05

Exercise 1 lec05

- How can we modify the code so to identify first match if there are duplicates?

search for 8

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]			
[	-3	0	4	6	8	8	11	]	lo = 0	m = 3
				A[m]					hi = 7	
[	-3	0	4	6	8	8	11	]	lo = 4	m = 5
						A[m]			hi = 7	

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x \leq A[m]$
    $hi \leftarrow m$
  else
    $lo \leftarrow m + 1$
if $hi < n \text{ and } A[hi] = x$
  return hi
else
  return $not\ found$

Exercise 1 lec05

- How can we modify the code so to identify first match if there are duplicates?

search for 8

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]			
[	-3	0	4	6	8	8	11	]	lo = 0	m = 3
				A[m]					hi = 7	
[	-3	0	4	6	8	8	11	]	lo = 4	m = 5
						A[m]			hi = 7	
[	-3	0	4	6	8	8	11	]	lo = 4	m = 4
					A[m]				hi = 5	

$lo, hi \leftarrow 0, n$
while $lo < hi$
    $m \leftarrow (lo + hi)/2$
    if $x \leq A[m]$
        $hi \leftarrow m$
    else
        $lo \leftarrow m + 1$
if $hi < n \; and \; A[hi] = x$
    return hi
else
    return $not\ found$

Exercise 1 lec05

➢ How can we modify the code so to identify first match if there are duplicates?

search for 8

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]			
[	-3	0	4	6	8	8	11	]	lo = 0	m = 3
				A[m]					hi = 7	
[	-3	0	4	6	8	8	11	]	lo = 4	m = 5
						A[m]			hi = 7	
[	-3	0	4	6	8	8	11	]	lo = 4	m = 4
					A[m]				hi = 5	
[	-3	0	4	6	8	8	11	]	lo = 4	
									hi = 4	

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x \leq A[m]$
    $hi \leftarrow m$
  else
    $lo \leftarrow m + 1$
if $hi < n$ *and* $A[hi] = x$
  return hi
else
  return *not found*

Exercise 1 lec05

- How can we modify the code so to identify first match if there are duplicates?

search for 8

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]			
[	-3	0	4	6	8	8	11	]	lo = 0	m = 3
				A[m]					hi = 7	
[	-3	0	4	6	8	8	11	]	lo = 4	m = 5
						A[m]			hi = 7	
[	-3	0	4	6	8	8	11	]	lo = 4	m = 4
					A[m]				hi = 5	
[	-3	0	4	6	8	8	11	]	lo = 4	
									hi = 4	

$lo, hi \leftarrow 0, n$
while $lo < hi$
  $m \leftarrow (lo + hi)/2$
  if $x \leq A[m]$
    $hi \leftarrow m$
  else
    $lo \leftarrow m + 1$
if $hi < n$ *and* $A[hi] = x$
  return hi
else
  return *not found*

Quicksort

Quicksort

quicksort

if $n \leq 1$
    return
else
    $p \leftarrow any\ element\ in\ A[0 \ \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0 \ \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

partition

while $next < fg$
    if $A[next] < p$
        swap $A[fe]$ and $A[next]$
        $fe, next \leftarrow fe + 1, next + 1$
    else if $A[next] > p$
        swap $A[next]$ and $A[fg - 1]$
        $fg \leftarrow fg - 1$
    else
        $next \leftarrow next + 1$

Quicksort

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	2	7	1	6	**5**

next =0

f_e = 0

f_g = 5

quicksort

if $n \leq 1$
  return
else
  $p \leftarrow$ *any element in* $A[0 \ldots n-1]$
  $(f_e, f_g) \leftarrow partition\ (A, n, p)$
  $quicksort\ (A[0 \ldots f_e - 1])$
  $quicksort\ (A[f_g \ldots n-1])$

partition

$next, fe, fg \leftarrow 0, 0, n$
while $next < fg$
  if $A[next] < p$
    swap $A[fe]$ and $A[next]$
    $fe, next \leftarrow fe + 1, next + 1$
  else if $A[next] > p$
    swap $A[next]$ and $A[fg - 1]$
    $fg \leftarrow fg - 1$
  else
    $next \leftarrow next + 1$

Quicksort

pivot: p = 5

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	
next =0	[**5**	2	7	1	6	**5**]	f_e = 0, f_g = 5
next=1	[**5**	2	7	1	6	**5**]	

quicksort

if $n \leq 1$

  return

else

  $p \leftarrow$ *any element in* $A[0 \; \ldots n-1]$

  $(f_e, f_g) \leftarrow partition\ (A, n, p)$

  $quicksort\ (A[0 \; \ldots f_e - 1])$

  $quicksort\ (A[f_g \ldots n-1])$

partition

$next, fe, fg \leftarrow 0, 0, n$

while $next < fg$

  if $A[next] < p$

    swap $A[fe]$ and $A[next]$

    $fe, next \leftarrow fe + 1, next + 1$

  else if $A[next] > p$

    swap $A[next]$ and $A[fg - 1]$

    $fg \leftarrow fg - 1$

  else

    $next \leftarrow next + 1$

Quicksort

pivot: p = 5

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	
next =0 [	$\boxed{5}$	2	7	1	6	$\boxed{5}$	]
next=1 [	$\boxed{5}$	$\boxed{2}$	7	1	6	$\boxed{5}$	]

$f_e = 0$

$f_g = 5$

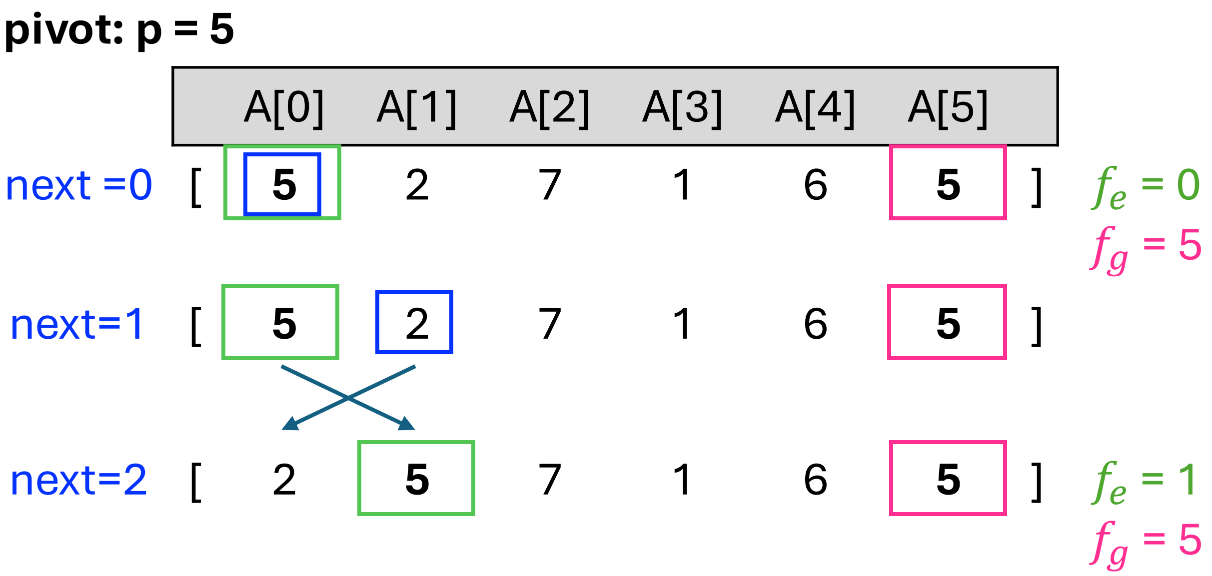
quicksort

if $n \leq 1$
  return
else
  $p \leftarrow$ *any element in* $A[0 \;\ldots n-1]$
  $(f_e, f_g) \leftarrow partition\ (A, n, p)$
  $quicksort\ (A[0 \;\ldots f_e - 1])$
  $quicksort\ (A[f_g \ldots n-1])$

partition

$next, fe, fg \leftarrow 0, 0, n$
while $next < fg$
  if $A[next] < p$
    swap $A[fe]$ and $A[next]$
    $fe, next \leftarrow fe + 1, next + 1$
  else if $A[next] > p$
    swap $A[next]$ and $A[fg - 1]$
    $fg \leftarrow fg - 1$
  else
    $next \leftarrow next + 1$

Quicksort



quicksort

if $n \leq 1$
    return
else
    $p \leftarrow$ *any element in* $A[0 \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0 \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

partition

$next, fe, fg \leftarrow 0, 0, n$
while $next < fg$
    if $A[next] < p$
        swap $A[fe]$ and $A[next]$
        $fe, next \leftarrow fe + 1, next + 1$
    else if $A[next] > p$
        swap $A[next]$ and $A[fg - 1]$
        $fg \leftarrow fg - 1$
    else
        $next \leftarrow next + 1$

Quicksort

pivot: p = 5

		A[0]	A[1]	A[2]	A[3]	A[4]	A[5]		
next =0	[	**5**	2	7	1	6	**5**	]	f_e = 0, f_g = 5
next=1	[	**5**	2	7	1	6	**5**	]	
next=2	[	2	**5**	7	1	6	**5**	]	f_e = 1, f_g = 5

quicksort

```
if n ≤ 1
    return
else
    p ← any element in A[0 ... n − 1]
    (f_e, f_g) ← partition (A, n, p)
    quicksort (A[0 ... f_e − 1])
    quicksort (A[f_g ... n − 1])
```

partition

```
next, fe, fg ← 0, 0, n
while next < fg
    if A[next] < p
        swap A[fe] and A[next]
        fe, next ← fe + 1, next + 1
    else if A[next] > p
        swap A[next] and A[fg − 1]
        fg ← fg − 1
    else
        next ← next + 1
```

Quicksort

pivot: p = 5

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	
next =0	**5**	2	7	1	6	**5**	f_e = 0, f_g = 5
next=1	**5**	2	7	1	6	**5**	
next=2	2	**5**	7	1	6	**5**	f_e = 1, f_g = 5
next=3	2	**5**	**5**	1	6	7	f_e = 1, f_g = 4

quicksort

if $n \leq 1$
    return
else
    $p \leftarrow$ *any element in* $A[0 \ \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0 \ \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

partition

$next, fe, fg \leftarrow 0, 0, n$
while $next < fg$
    if $A[next] < p$
        swap $A[fe]$ and $A[next]$
        $fe, next \leftarrow fe + 1, next + 1$
    else if $A[next] > p$
        swap $A[next]$ and $A[fg - 1]$
        $fg \leftarrow fg - 1$
    else
        $next \leftarrow next + 1$

Quicksort

pivot: p = 5

		A[0]	A[1]	A[2]	A[3]	A[4]	A[5]		
next =0	[	**5**	2	7	1	6	**5**	]	f_e = 0, f_g = 5
next=1	[	**5**	2	7	1	6	**5**	]	
next=2	[	2	**5**	7	1	6	**5**	]	f_e = 1, f_g = 5
next=3	[	2	**5**	**5**	1	6	7	]	f_e = 1, f_g = 4

quicksort

if $n \leq 1$
    return
else
    $p \leftarrow$ *any element in* $A[0 \; \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0 \; \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

partition

next, *fe*, *fg* $\leftarrow 0, 0, n$
while *next* $<$ *fg*
    if $A[next] < p$
        swap $A[fe]$ and $A[next]$
        fe, *next* $\leftarrow fe + 1, next + 1$
    else if $A[next] > p$
        swap $A[next]$ and $A[fg - 1]$
        $fg \leftarrow fg - 1$
    else
        $next \leftarrow next + 1$

Quicksort

pivot: p = 5

		A[0]	A[1]	A[2]	A[3]	A[4]	A[5]		
next =0	[	**5**	2	7	1	6	**5**	]	f_e = 0 f_g = 5
next=1	[	**5**	2	7	1	6	**5**	]	
next=2	[	2	**5**	7	1	6	**5**	]	f_e = 1 f_g = 5
next=3	[	2	**5**	**5**	1	6	7	]	f_e = 1 f_g = 4

quicksort

if $n \leq 1$
    return
else
    $p \leftarrow any\ element\ in\ A[0\ \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0\ \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

partition

$next, fe, fg \leftarrow 0, 0, n$
while $next < fg$
    if $A[next] < p$
        swap $A[fe]$ and $A[next]$
        $fe, next \leftarrow fe + 1, next + 1$
    else if $A[next] > p$
        swap $A[next]$ and $A[fg - 1]$
        $fg \leftarrow fg - 1$
    else
        $next \leftarrow next + 1$

Quicksort

pivot: p = 5

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	
next =0	**5**	2	7	1	6	**5**	f_e = 0, f_g = 5
next=1	**5**	2	7	1	6	**5**	
next=2	2	**5**	7	1	6	**5**	f_e = 1, f_g = 5
next=3	2	**5**	**5**	1	6	7	f_e = 1, f_g = 4
next=3	2	1	**5**	**5**	6	7	f_e = 2, f_g = 4

quicksort

if $n \leq 1$
  return
else
  $p \leftarrow any\ element\ in\ A[0\ \ldots n-1]$
  $(f_e, f_g) \leftarrow partition\ (A, n, p)$
  $quicksort\ (A[0\ \ldots f_e - 1])$
  $quicksort\ (A[f_g \ldots n-1])$

partition

$next, fe, fg \leftarrow 0, 0, n$
while $next < fg$
  if $A[next] < p$
    swap $A[fe]$ and $A[next]$
    $fe, next \leftarrow fe + 1, next + 1$
  else if $A[next] > p$
    swap $A[next]$ and $A[fg - 1]$
    $fg \leftarrow fg - 1$
  else
    $next \leftarrow next + 1$

Quicksort

pivot: p = 5

		A[0]	A[1]	A[2]	A[3]	A[4]	A[5]		
next =0	[	**5**	2	7	1	6	**5**	]	f_e = 0 f_g = 5
next=1	[	**5**	2	7	1	6	**5**	]	
next=2	[	2	**5**	7	1	6	**5**	]	f_e = 1 f_g = 5
next=3	[	2	**5**	**5**	1	6	7	]	f_e = 1 f_g = 4
next=4	[	2	1	**5**	**5**	6	7	]	f_e = 2 f_g = 4

quicksort

if $n \leq 1$

  return

else

  $p \leftarrow any\ element\ in\ A[0\ \ldots n-1]$

  $(f_e, f_g) \leftarrow partition\ (A, n, p)$

  $quicksort\ (A[0\ \ldots f_e - 1])$

  $quicksort\ (A[f_g \ldots n-1])$

partition

$next, fe, fg \leftarrow 0, 0, n$

while $next < fg$

  if $A[next] < p$

    swap $A[fe]$ and $A[next]$

    $fe, next \leftarrow fe + 1, next + 1$

  else if $A[next] > p$

    swap $A[next]$ and $A[fg - 1]$

    $fg \leftarrow fg - 1$

  else

    $next \leftarrow next + 1$

Quicksort

pivot: p = 5

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	
next =0	**5**	2	7	1	6	**5**	f_e = 0, f_g = 5
next=1	**5**	2	7	1	6	**5**	
next=2	2	**5**	7	1	6	**5**	f_e = 1, f_g = 5
next=3	2	**5**	**5**	1	6	7	f_e = 1, f_g = 4
next=4	2	1	**5**	**5**	6	7	f_e = 2, f_g = 4

smaller than pivot · larger than pivot

pivot in correct position 🎉

quicksort

if $n \le 1$
  return
else
  $p \leftarrow \textit{any element in } A[0 \ldots n-1]$
  $(f_e, f_g) \leftarrow partition\,(A, n, p)$
  $quicksort\,(A[0 \ldots f_e - 1])$
  $quicksort\,(A[f_g \ldots n-1])$

partition

$next, fe, fg \leftarrow 0, 0, n$
while $next < fg$
  if $A[next] < p$
    swap $A[fe]$ and $A[next]$
    $fe, next \leftarrow fe + 1, next + 1$
  else if $A[next] > p$
    swap $A[next]$ and $A[fg - 1]$
    $fg \leftarrow fg - 1$
  else
    $next \leftarrow next + 1$

Quicksort

quicksort

if $n \leq 1$
  return
else
  $p \leftarrow any\ element\ in\ A[0\ \ldots n-1]$
  $(f_e, f_g) \leftarrow partition\ (A, n, p)$
  $quicksort\ (A[0\ \ldots f_e - 1])$
  $quicksort\ (A[f_g \ldots n-1])$

partition

while $next < fg$
  if $A[next] < p$
    swap $A[fe]$ and $A[next]$
    $fe, next \leftarrow fe + 1, next + 1$
  else if $A[next] > p$
    swap $A[next]$ and $A[fg - 1]$
    $fg \leftarrow fg - 1$
  else
    $next \leftarrow next + 1$

Case	Time	Example
Best		
Average		
Worst		

Quicksort

quicksort

if $n \leq 1$
    return
else
    $p \leftarrow any\ element\ in\ A[0\ \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0\ \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

partition

while $next < fg$
    if $A[next] < p$
        swap $A[fe]$ and $A[next]$
        $fe, next \leftarrow fe + 1, next + 1$
    else if $A[next] > p$
        swap $A[next]$ and $A[fg - 1]$
        $fg \leftarrow fg - 1$
    else
        $next \leftarrow next + 1$

Case	Time	Example
Best	$O(n)$	[1,1,1,1,1,1,1]
Average	$O(n \log n)$	[1, 3, 6, 2, 4, 5, 7] picking a pivot that splits the array approximately in half
Worst	$O(n^2)$	[1, 3, 6, 2, 4, 5, 7] picking a pivot that makes a partition of size $n - 1$

Exercise 4 lec05

Exercise 4 lec05

A slightly different approach to *partition* is described by this more relaxed specification:

assert: $n > 1$ **and** $p \in A[0 \ldots n-1]$
$f \leftarrow \mathit{partition}(A, n, p)$
assert: $0 \leq f \leq n-1$ **and** $A[0 \ldots f-1] \leq p$ **and**
$A[f] = p$ **and** $A[f+1 \ldots n-1] \geq p$

4a. Design a function that meets this specification.

4b. Does this approach have any disadvantages or advantages compared to the first one presented?

Exercise 4 lec05

A slightly different approach to *partition* is described by this more relaxed specification:

assert: $n > 1$ **and** $p \in A[0 \ldots n-1]$
$f \leftarrow partition(A, n, p)$
assert: $0 \leq f \leq n-1$ **and** $A[0 \ldots f-1] \leq p$ **and**
$A[f] = p$ **and** $A[f+1 \ldots n-1] \geq p$

(Previous specification)

assert: $n > 1$ **and** $p \in A[0 \ldots n-1]$
$(fe, fg) \leftarrow partition(A, n, p)$
assert: $0 \leq fe < fg \leq n$ **and** $A[0 \ldots fe-1] < p$ **and**
$A[fe \ldots fg-1] = p$ **and** $A[fg \ldots n-1] > p$

Slide 37 of lec05

4a. Design a function that meets this specification.

4b. Does this approach have any disadvantages or advantages compared to the first one presented?

Slide 55 of lec05

Exercise 4 lec05

A slightly different approach to *partition* is described by this more relaxed specification:

assert: $n > 1$ **and** $p \in A[0 \ldots n-1]$
$f \leftarrow partition(A, n, p)$
assert: $0 \leq f \leq n-1$ **and** $A[0 \ldots f-1] \leq p$ **and**
$A[f] = p$ **and** $A[f+1 \ldots n-1] \geq p$

(Previous specification)

assert: $n > 1$ **and** $p \in A[0 \ldots n-1]$
$(fe, fg) \leftarrow partition(A, n, p)$
assert: $0 \leq fe < fg \leq n$ **and** $A[0 \ldots fe-1] < p$ **and**
$A[fe \ldots fg-1] = p$ **and** $A[fg \ldots n-1] > p$

Slide 37 of lec05

4a. Design a function that meets this specification.

4b. Does this approach have any disadvantages or advantages compared to the first one presented?

Slide 55 of lec05

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	2	7	1	6	5

[5 2 7 1 6 5]

itemFromLeft = 1. item from the left that is larger than the pivot

itemFromRight = 1. item from the left that is larger than the pivot

- ➢ swap pivot to the left

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	2	7	1	6	5

itemFromLeft = 1. item from the left that is larger than the pivot

itemFromRight = 1. item from the left that is larger than the pivot

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	2	7	1	6	5

itemFromLeft = 1. item from the left that is larger than the pivot

itemFromRight = 1. item from the left that is larger than the pivot

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	2	$\boxed{7}$	1	6	5

itemFromLeft = 1. item from the left that is larger than the pivot

itemFromRight = 1. item from the left that is larger than the pivot

Exercise 4 lec05

pivot: p = 5

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	
[	5	2	$\boxed{7}$	1	6	5	]

itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

Exercise 4 lec05

pivot: p = 5

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	
[	5	2	$\boxed{7}$	1	6	$\boxed{5}$	]

itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	2	7	1	6	5
5	2	5	1	6	7

itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	2	7	1	6	5
5	2	5	1	6	7

itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
[5	2	7	1	6	5]
[5	2	5	1	6	7]

itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	2	7	1	6	5
5	2	5	1	6	7

itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]

[5 2 7 1 6 5]

[5 2 5 1 $\boxed{6}$ 7]

itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	2	7	1	6	5
5	2	5	1	6	7

itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

Exercise 4 lec05

pivot: p = 5

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	
[	5	2	7	1	6	5	]
[	5	2	5	1	$\boxed{6}$	7	]

itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	2	7	1	6	5
5	2	5	$\boxed{1}$	$\boxed{6}$	7

itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

- if itemFromRight comes before itemFromLeft we are done and swap the pivot back

Exercise 4 lec05

pivot: p = 5

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	
[	5	2	7	1	6	5	]
[	5	2	5	$\boxed{1}$	$\boxed{6}$	7	]
[	1	2	5	5	6	7	]

itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

- if itemFromRight comes before itemFromLeft we are done and swap the pivot back

Exercise 4 lec05

pivot: p = 5

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]

[5 2 7 1 6 5]

[5 2 5 1 6 7]

[1 2 5 5 6 7]

$\underbrace{1 \quad 2 \quad 5}_{\text{smaller than pivot}} \quad 5 \quad \underbrace{6 \quad 7}_{\text{larger than pivot}}$

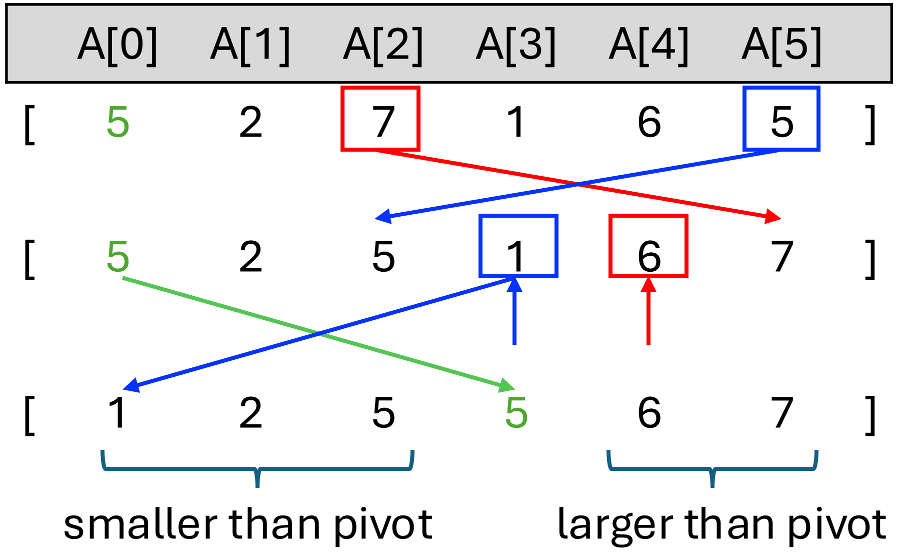
itemFromLeft = 1. item from the left that is larger than the pivot (or equal)

itemFromRight = 1. item from the right that is smaller than the pivot (or equal)

➢ if itemFromRight comes before itemFromLeft we are done and swap the pivot back

Exercise 4 lec05

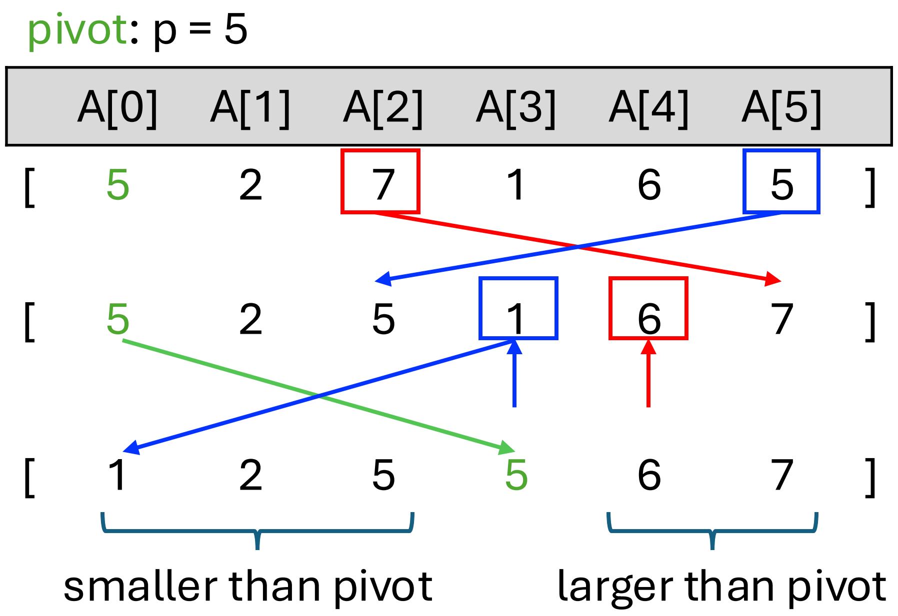
pivot: p = 5



pseudocode

$identify\ x\ such\ that\ A[x] = p$
$swap\ (A[0], A[x])$
$le, ge \leftarrow 1, n-1$ *// le = less than or equal (itemFromRight), ge = greater than or equal (itemFromLeft)*

Exercise 4 lec05



pseudocode

$identify\ x\ such\ that\ A[x] = p$

$swap\ (A[0], A[x])$

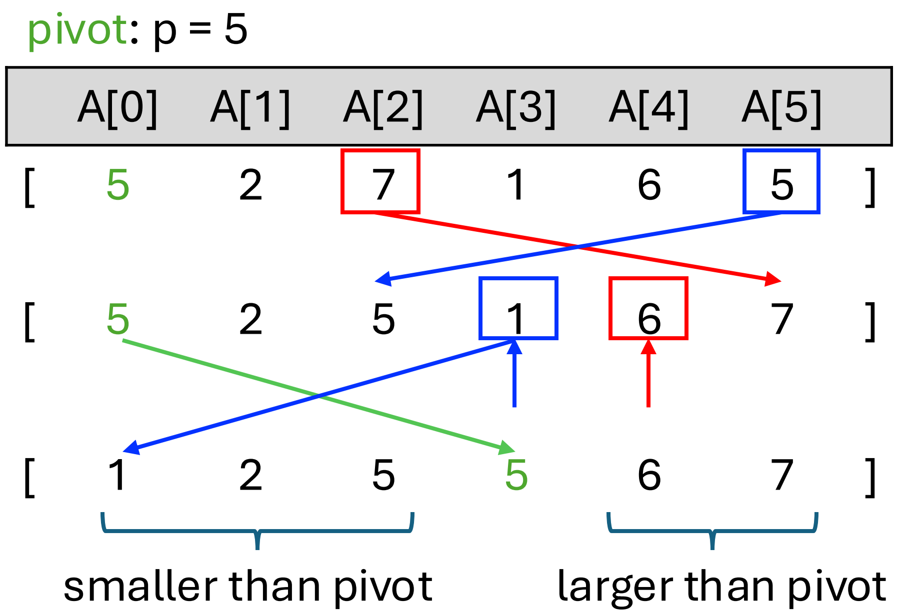
$le, ge \leftarrow 1, n-1$ // le = less than or equal (*itemFromRight*), ge = greater than or equal (*itemFromLeft*)

while $le \leq ge$

    while $le < n$ **and** $A[le] < p$

        $le \leftarrow le + 1$

Exercise 4 lec05

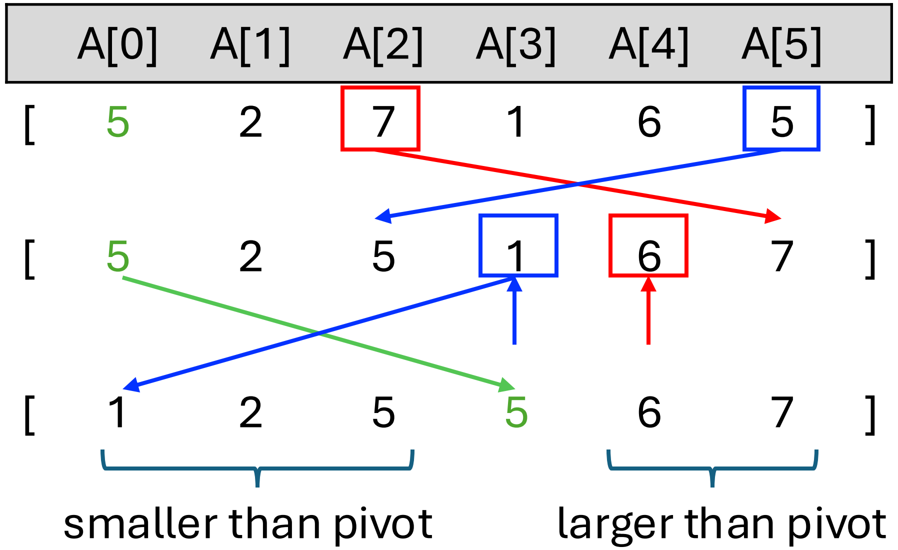


pseudocode

$identify\ \ x\ \ such\ that\ A[x] = p$
$swap\ (A[0], A[x])$
$le, ge\ \leftarrow 1, n-1$ *// le = less than or equal (itemFromRight), ge = greater than or equal (itemFromLeft)*
while $le \leq ge$
    while $le < n$ **and** $A[le] < p$
        $le\ \leftarrow le + 1$
    while $A[ge] > p$
        $ge\ \leftarrow ge - 1$

Exercise 4 lec05

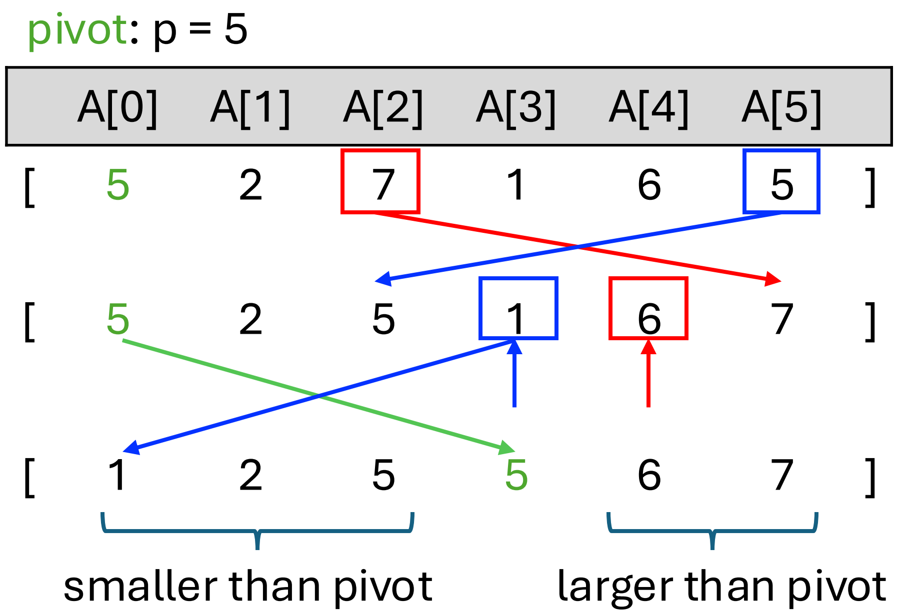
pivot: p = 5



pseudocode

```
identify x such that A[x] = p
swap (A[0], A[x])
le, ge ← 1, n − 1   // le = less than or equal (itemFromRight),
                       ge = greater than or equal (itemFromLeft)
while le ≤ ge
    while le < n and A[le] < p
        le ← le + 1
    while A[ge] > p
        ge ← ge − 1
    if le ≤ ge
        swap(A[le], A[ge])
        le, ge ← le + 1, ge − 1
```

Exercise 4 lec05



pseudocode

$identify\ \ x\ \ such\ that\ A[x] = p$

$swap\ (A[0], A[x])$

$le, ge\ \leftarrow 1, n-1$ // le = less than or equal (itemFromRight), ge = greater than or equal (itemFromLeft)

while $le \leq ge$

  while $le < n$ **and** $A[le] < p$

    $le\ \leftarrow le + 1$

  while $A[ge] > p$

    $ge\ \leftarrow ge - 1$

  if $le \leq ge$

    $swap(A[le], A[ge])$

    $le, ge\ \leftarrow le + 1, ge - 1$

$swap(A[0], A[ge])$

return ge

Exercise 4 lec05

assert: $n > 1$ **and** $p \in A[0 \ldots n-1]$
$f \leftarrow partition(A, n, p)$
assert: $0 \leq f \leq n-1$ **and** $A[0 \ldots f-1] \leq p$ **and**
$A[f] = p$ **and** $A[f+1 \ldots n-1] \geq p$

pseudocode

$identify\ \ x\ \ such\ that\ A[x] = p$
$swap\ (A[0], A[x])$
$le, ge\ \leftarrow 1, n-1$ // le = less than or equal (itemFromRight), ge = greater than or equal (itemFromLeft)
while $le \leq ge$
    while $le < n$ **and** $A[le] < p$
        $le\ \leftarrow le + 1$
    while $A[ge] > p$
        $ge\ \leftarrow ge - 1$
    if $le \leq ge$
        $swap(A[le], A[ge])$
        $le, ge\ \leftarrow le + 1, ge - 1$
$swap(A[0], A[ge])$
return ge

Exercise 4 lec05

Advantages:

- Fewer element swaps than the three-way partitioning

Disadvantages:

- Still leads to $\mathrm{O}(n \log n)$ time on arrays with repeated elements whereas three-way partitioning gives $\mathrm{O}(n)$ time on arrays and sub-arrays in which all n items are the same

Case	Time	Example
Best	$O(n \log n)$	[1,1,1,1,1,1,1]
Average	$O(n \log n)$	[1, 3, 6, 2, 4, 5, 7] picking a pivot that splits the array approximately in half
Worst	$O(n^2)$	[1, 3, 6, 2, 4, 5, 7] picking a pivot that makes a partition of size $n - 1$