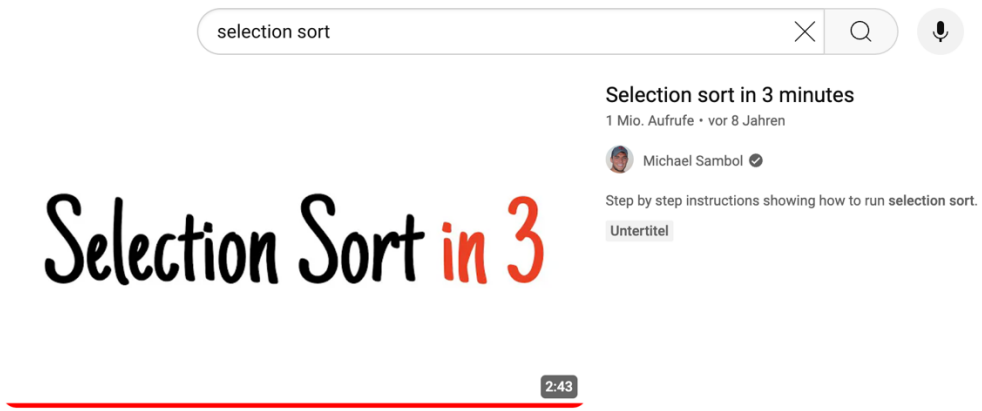
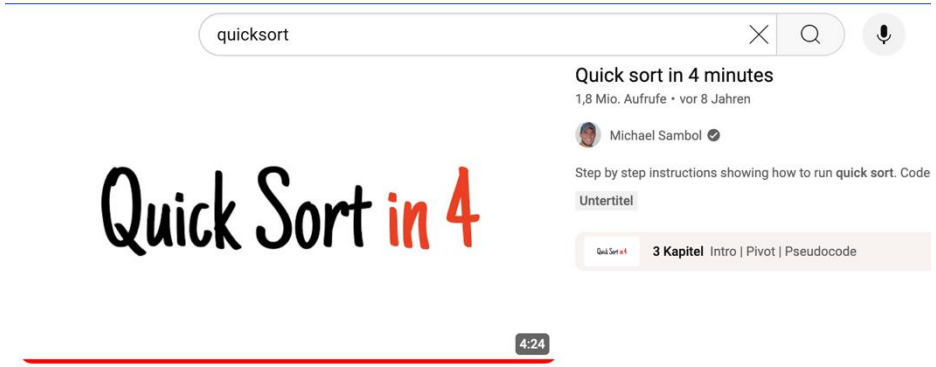
Hello Class!

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

1) https://www.youtube.com/watch?v=JU767SDMDvA
2) https://www.youtube.com/watch?v=g-PGLbMth_g&t=3s
3) https://www.youtube.com/watch?v=Hoixgm4-P4M





Tips for the MST

- Do **all past tests** — they repeat patterns!
- Do the **sample test** (this year's) **twice**

⚠️ Common coding mistakes:

- **Integer division:** `5/2` will return 2, **not** 2.5!
  Use `5.0/2.0`, or cast: `(double)5/2`

- **Loop variable not declared**
  - `for (int i=0; i<n; i++)` ✅ don't forget `int`

- **For non-void functions:**
  - Make sure you **return something** in **every possible path**!

Strings in C

- A **string in C** is an **array of characters ending with \0** (null terminator)
  `char str[] = "Hello"` → stored as `'H' 'e' 'l' 'l' 'o' '\0'`
- ⚠ Always ensure enough space for `\0`!

- Use `<string.h>` for string functions:
  - `strlen(str)` – length (excludes `\0`)
  - `strcpy(dest, src)` – copy
  - `strcmp(str1, str2)` – compare
  - `strcat(dest, src)` – concatenate

- **String** vs **String literal**:

Type	Example	Writable?
String literal	`char *s = "Hello"`	❌ read-only
String	`char s[] = "Hello"`	✅ writable

Quicksort

quicksort

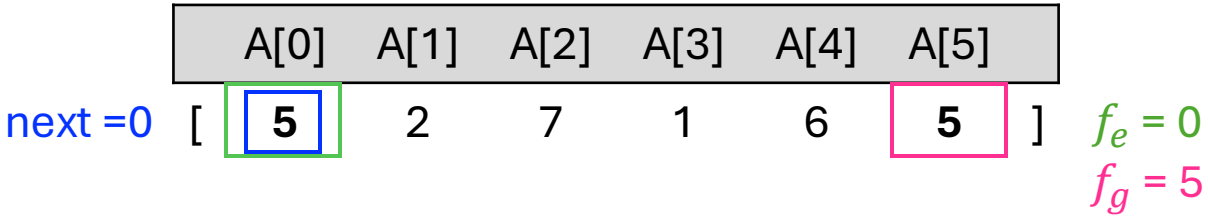
if $n \leq 1$
    return
else
    $p \leftarrow any\ element\ in\ A[0\ \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0\ \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

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



quicksort

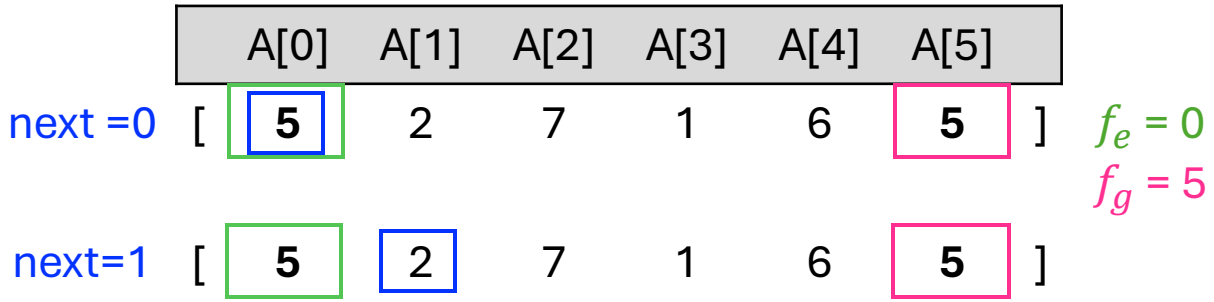
if $n \leq 1$
    return
else
    $p \leftarrow$ *any element in* $A[0 \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0 \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

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



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

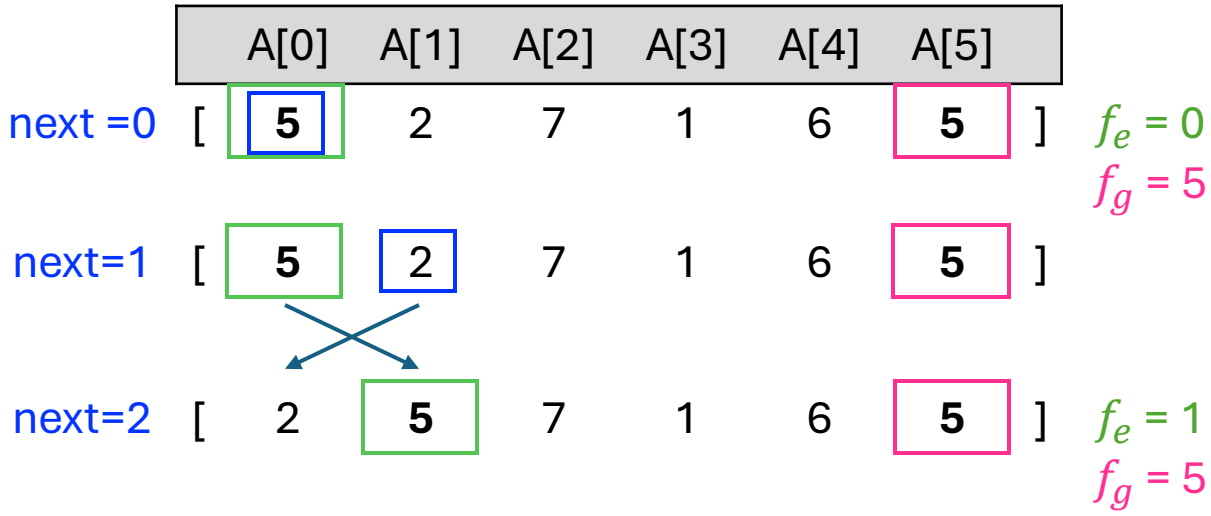
if $n \leq 1$
    return
else
    $p \leftarrow$ *any element in* $A[0 \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0 \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

partition

```
next, fe, fg ← 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



quicksort

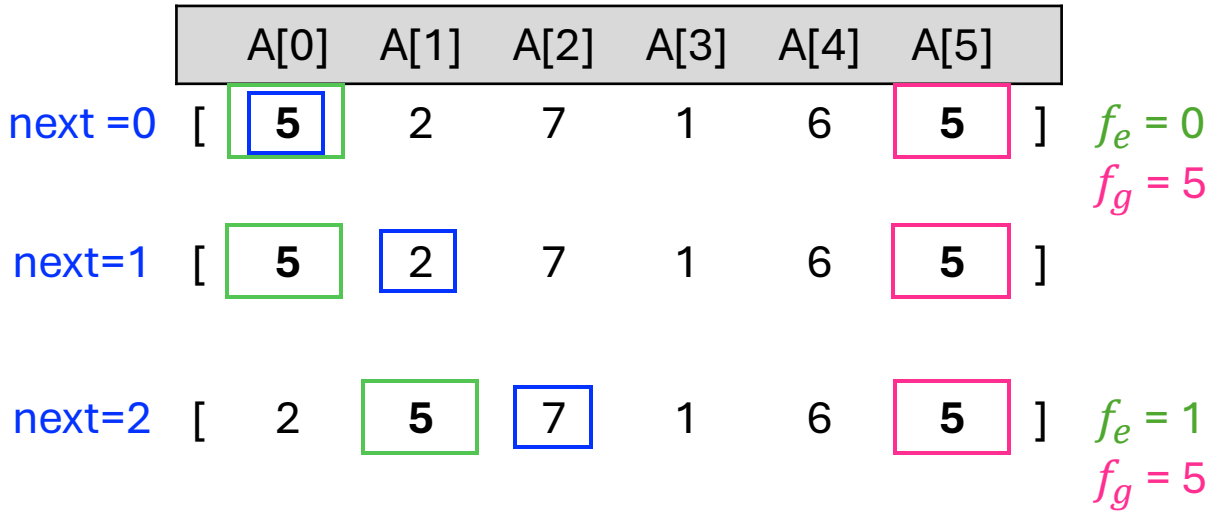
if $n \leq 1$
    return
else
    $p \leftarrow$ *any element in* $A[0 \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0 \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

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



quicksort

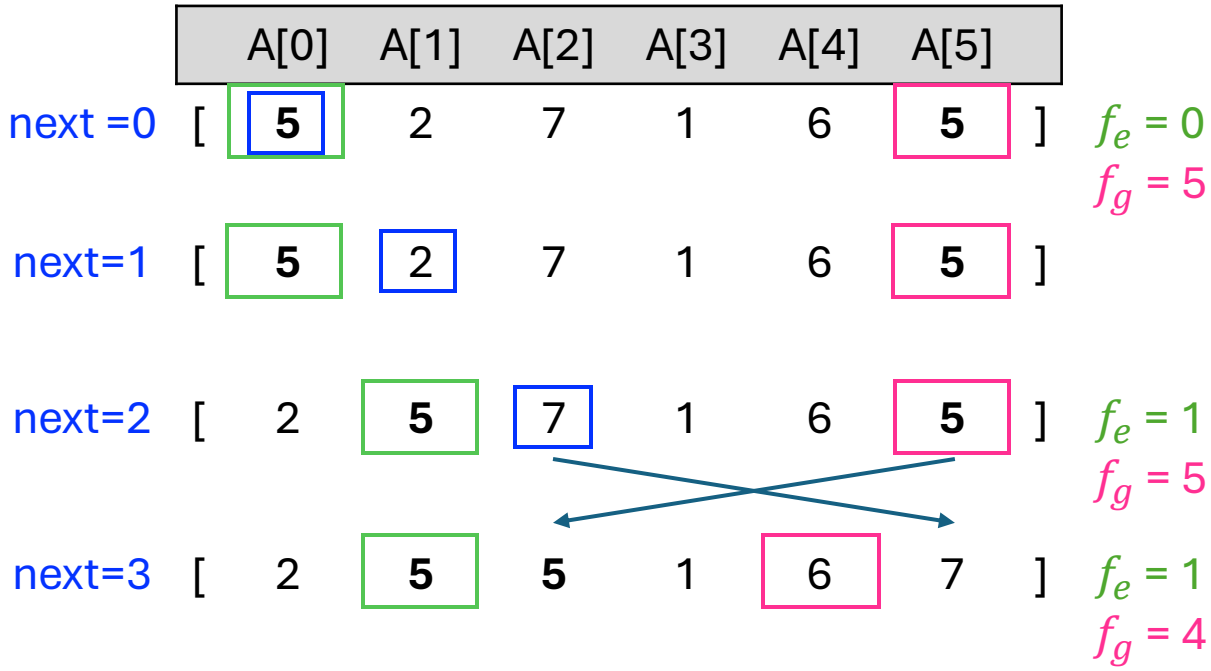
if $n \leq 1$
  return
else
  $p \leftarrow any\ element\ in\ A[0\ \ldots n-1]$
  $(f_e, f_g) \leftarrow partition\ (A, n, p)$
  $quicksort\ (A[0\ \ldots f_e - 1])$
  $quicksort\ (A[f_g \ldots n-1])$

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



quicksort

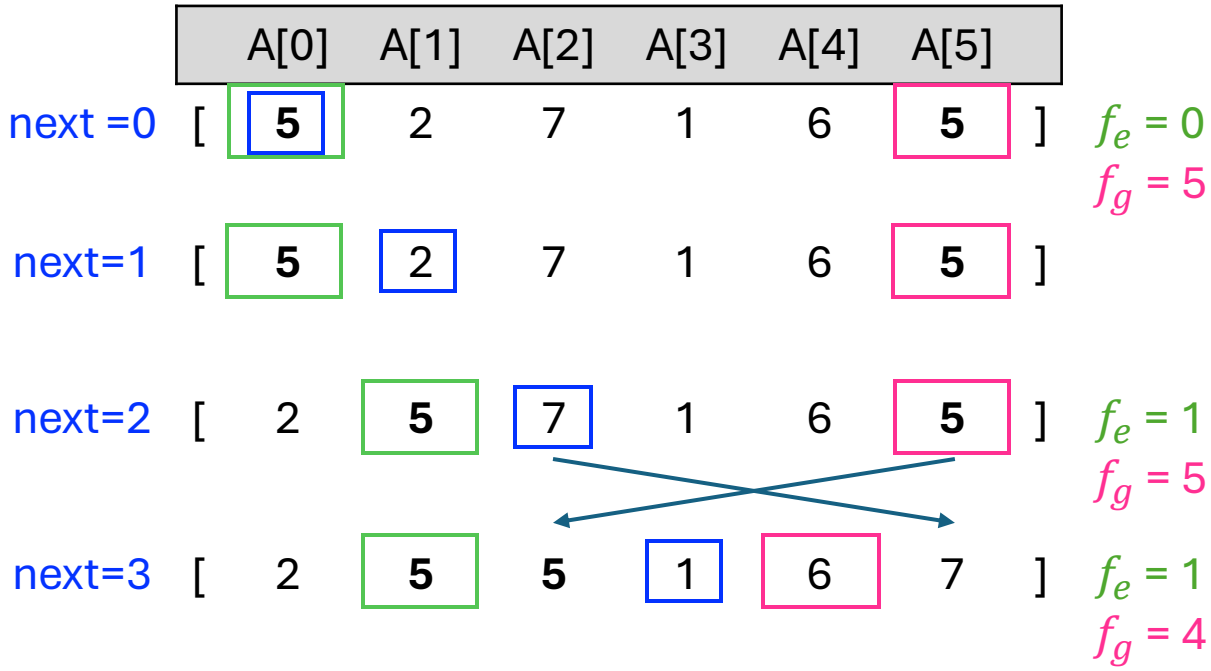
if $n \leq 1$
    return
else
    $p \leftarrow$ *any element in* $A[0 \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0 \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

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



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

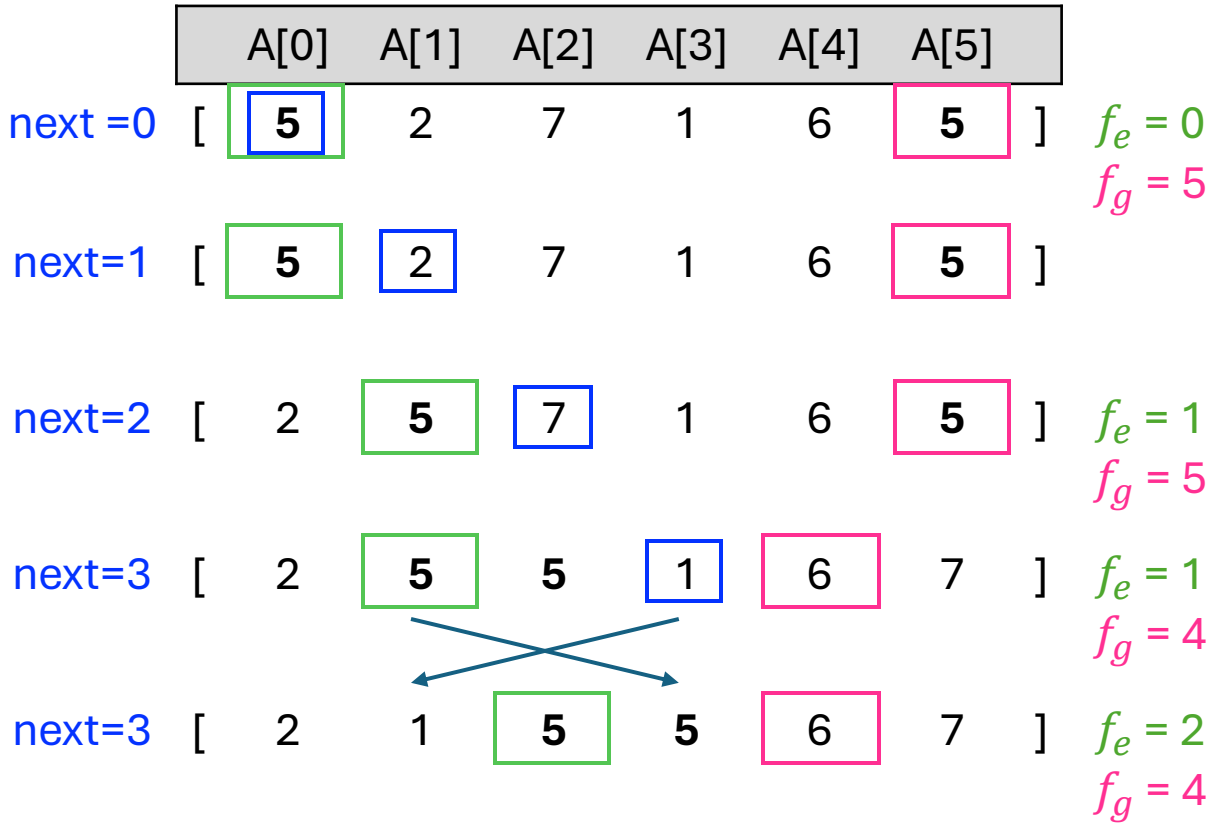
if $n \leq 1$
    return
else
    $p \leftarrow$ *any element in* $A[0 \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0 \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

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



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

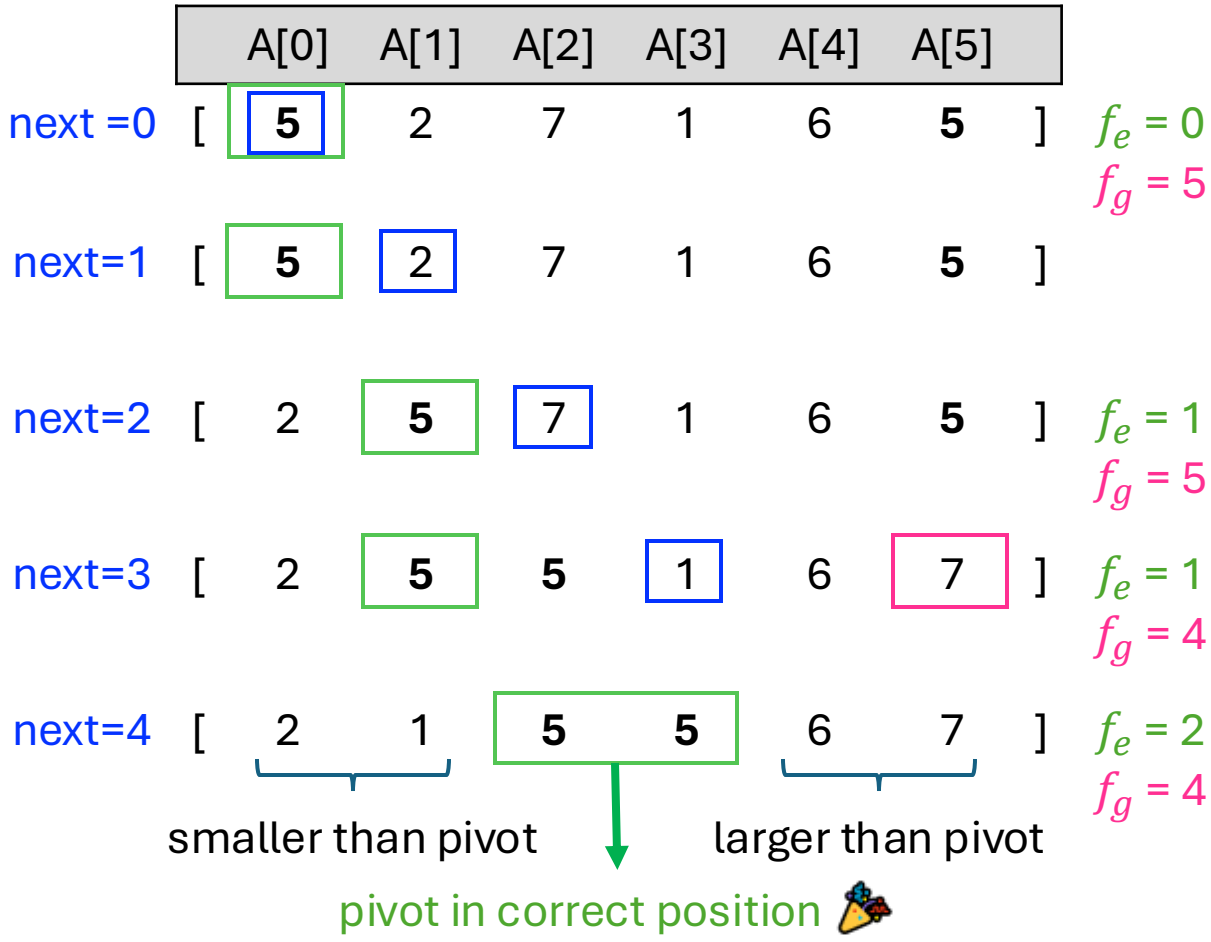
if $n \leq 1$
    return
else
    $p \leftarrow$ *any element in* $A[0 \ldots n-1]$
    $(f_e, f_g) \leftarrow partition\ (A, n, p)$
    $quicksort\ (A[0 \ldots f_e - 1])$
    $quicksort\ (A[f_g \ldots n-1])$

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



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 ≤ 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

```
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
```

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 $n - 1$

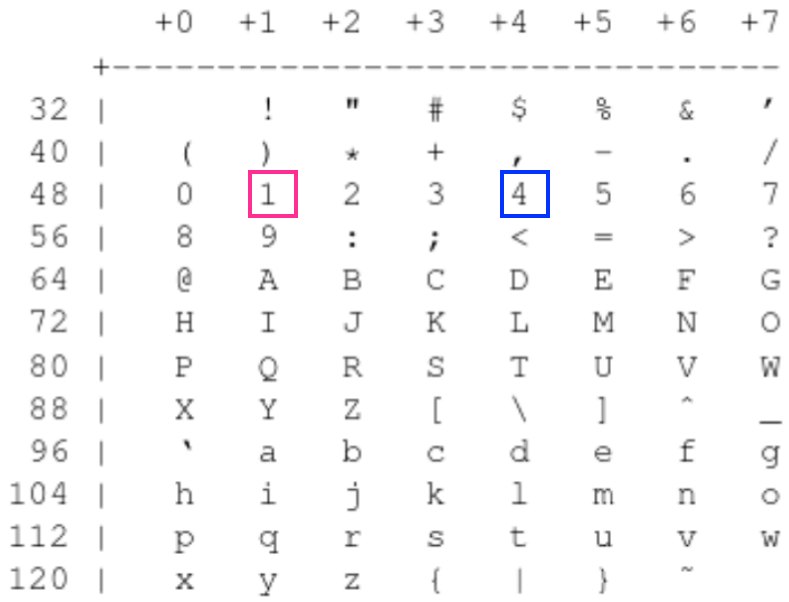
Converting `str` to `int`

Sample outputs

```
    45
The string "    45" corresponds to integer 45
rt45
The string "rt45" corresponds to integer 0
45yhu
The string "45yhu" corresponds to integer 45

The string "" corresponds to integer 0
 4 5
The string " 4 5" corresponds to integer 4
```

Converting str to int using ASCII values



	+0	+1	+2	+3	+4	+5	+6	+7
32		!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7
56	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G
72	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W
88	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g
104	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	

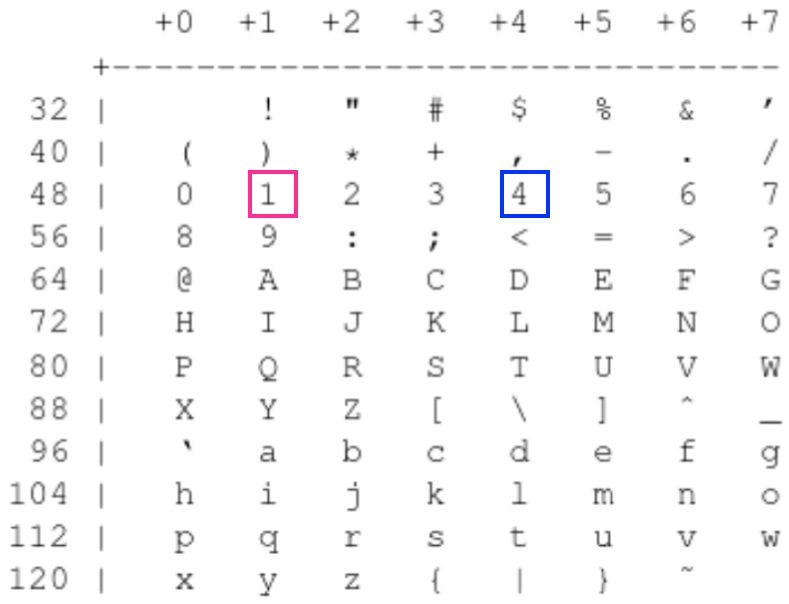
Converting `str` to `int`

Sample outputs

```
    45
The string "    45" corresponds to integer 45
rt45
The string "rt45" corresponds to integer 0
45yhu
The string "45yhu" corresponds to integer 45

The string "" corresponds to integer 0
 4 5
The string " 4 5" corresponds to integer 4
```

Converting str to int using ASCII values



	+0	+1	+2	+3	+4	+5	+6	+7
32		!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7
56	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G
72	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W
88	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g
104	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	

The ASCII code for the character ‘0’ is 48.

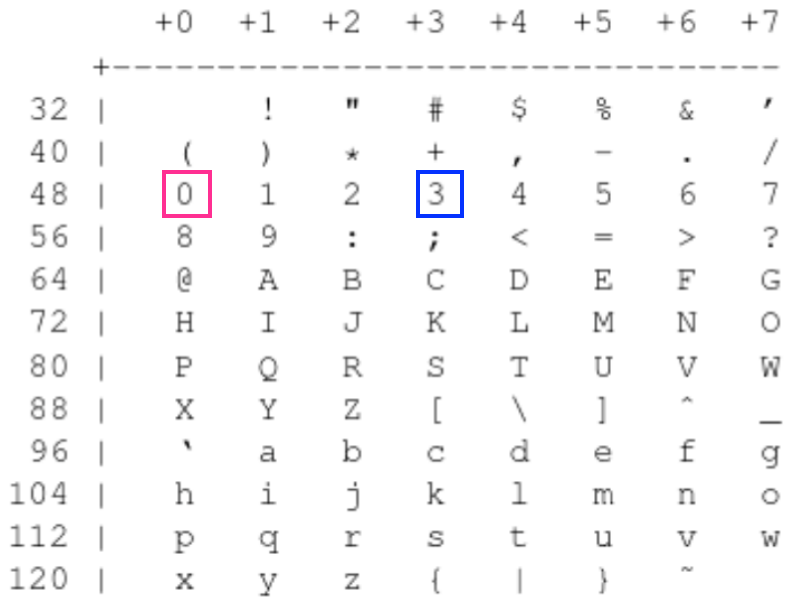
Converting `str` to `int`

Sample outputs

```
    45
The string "    45" corresponds to integer 45
rt45
The string "rt45" corresponds to integer 0
45yhu
The string "45yhu" corresponds to integer 45

The string "" corresponds to integer 0
 4 5
The string " 4 5" corresponds to integer 4
```

Converting str to int using ASCII values



	+0	+1	+2	+3	+4	+5	+6	+7
32		!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7
56	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G
72	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W
88	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g
104	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	

The ASCII code for the character ‘0’ is 48.
The ASCII code for the character ‘3’ is 51.

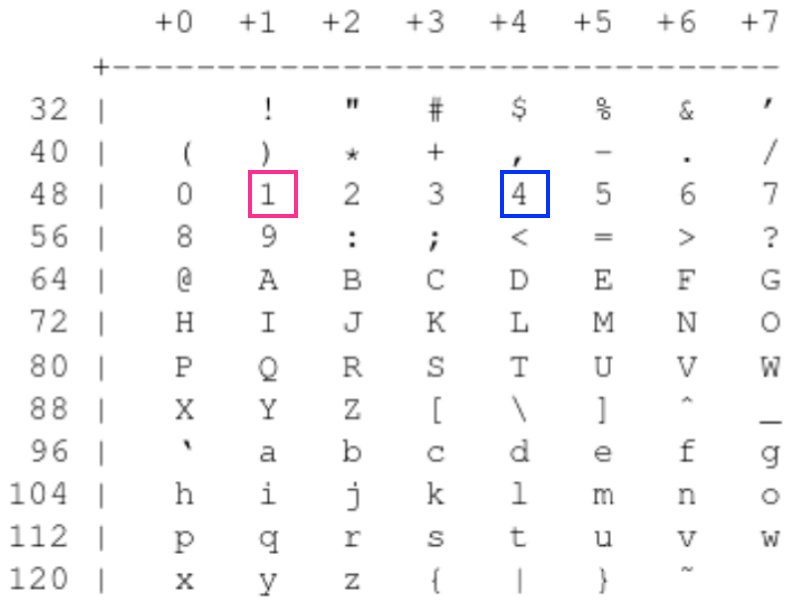
Converting `str` to `int`

Sample outputs

```
    45
The string "    45" corresponds to integer 45
rt45
The string "rt45" corresponds to integer 0
45yhu
The string "45yhu" corresponds to integer 45

The string "" corresponds to integer 0
 4 5
The string " 4 5" corresponds to integer 4
```

Converting str to int using ASCII values



	+0	+1	+2	+3	+4	+5	+6	+7
32		!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7
56	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G
72	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W
88	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g
104	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	

The ASCII code for the character ‘0’ is 48.
The ASCII code for the character ‘3’ is 51.

⇒ To get the corresponding integer, always subtract 48.

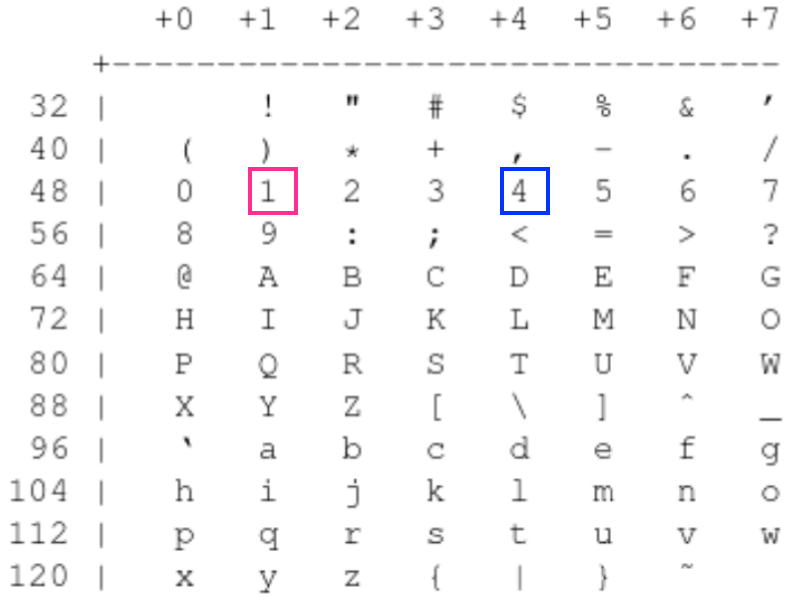
Converting `str` to `int`

Sample outputs

```
    45
The string "    45" corresponds to integer 45
rt45
The string "rt45" corresponds to integer 0
45yhu
The string "45yhu" corresponds to integer 45

The string "" corresponds to integer 0
 4 5
The string " 4 5" corresponds to integer 4
```

Converting str to int using ASCII values



	+0	+1	+2	+3	+4	+5	+6	+7
32		!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7
56	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G
72	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W
88	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g
104	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	

The ASCII code for the character '0' is 48. 48-48 = 0 ✓

The ASCII code for the character '3' is 51. 51-48 = 3 ✓

⇒ To get the corresponding integer, always subtract 48.