

Hello Class!

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

[https://www.youtube.com/watch?v=g-PGLbMth\\_g](https://www.youtube.com/watch?v=g-PGLbMth_g)

The screenshot shows the YouTube interface. On the left is the sidebar with the YouTube logo and navigation links: Startseite, Shorts, Abos, and a 'Mein YouTube' section with links to 'Mein Kanal', 'Verlauf', and 'Playlists'. The top navigation bar includes a search bar with the text 'selection sort', a search icon, a microphone icon, and user avatars for a plus icon, a notification bell with '9+', and a user profile 'M'. The video player area displays the title 'Selection Sort in 3' in a large, stylized font, with 'in 3' in red. Below the title is a red progress bar and a timestamp '2:43'. To the right of the video, the video title 'Selection sort in 3 minutes' is shown, followed by '1 Mio. Aufrufe • vor 8 Jahren', the channel name 'Michael Sambol' with a verified badge, and a description: 'Step by step instructions showing how to run selection sort.' Below the description is a 'Untertitel' button. A vertical ellipsis menu is visible on the far right.

## Time Complexity

- Time complexity tells us **how long an algorithm takes depending on how big the input is**

Example 1: Checking **every student** in class

10 students → 10 checks ⚡ ⚡ ⚡ ⚡ ⚡  
1,000 students → 1,000 checks ⚡ ⚡ ⚡ ⚡

Number of steps **grows linearly**  $O(n)$

“Big-O notation”

## Time Complexity

- Time complexity tells us **how long an algorithm takes depending on how big the input is**

Example 1: Checking **every student** in class

10 students → 10 checks ⚡ ⚡ ⚡ ⚡ ⚡  
1,000 students → 1,000 checks ⚡ ⚡ ⚡ ⚡

Number of steps **grows linearly**  $O(n)$

“Big-O notation”

Example 2: Checking **every pair** of students

10 students → 100 checks ⚡ ⚡ ⚡ ⚡  
1,000 students → 1,000,000 checks ⚡

Number of steps **grows quadratically**  $O(n^2)$

## Time Complexity

- Time complexity tells us **how long an algorithm takes depending on how big the input is**

Example 1: Checking **every student** in class

10 students → 10 checks ⚡ ⚡ ⚡ ⚡ ⚡  
1,000 students → 1,000 checks ⚡ ⚡ ⚡ ⚡

Number of steps **grows linearly**  $O(n)$

“Big-O notation”

Example 2: Checking **every pair** of students

10 students → 100 checks ⚡ ⚡ ⚡ ⚡  
1,000 students → 1,000,000 checks ⚡

Number of steps **grows quadratically**  $O(n^2)$

$O(n)$	worst case
$\Omega(n)$	best case
$\Theta(n)$	average case

## Arrays

	<table><tr><td>A[0]</td><td>A[1]</td><td>A[2]</td><td>A[3]</td></tr></table>				A[0]	A[1]	A[2]	A[3]
A[0]	A[1]	A[2]	A[3]					
Array:	[	5	7	3	4	]		
Memory:		0xFF120	0xFF124	0xFF128	0xFF132			

- an array is a collection of same-type variables
- in C, an array is a sequential block of memory → you must define the size at creation and can't change it later on


## Initialising arrays

type of the array

shape of the array

access the  $i$ -th  
entry of the array

```
2  /* Initialise directly */
3
4  int A[3] = {1, 2, 3};
5
6  int A[5] = {1, 2}; // A[2] ... A[4] automatically set to 0
7
8  int A[2][3] = {      // 2d array
9      {1, 2, 3},
10     {4, 5, 6}
11 }
12
13
14 /* Fill up with a loop */
15
16 int A[3];
17
18 for (int i = 0; i < 3; i++) {
19     A[i] = i + 1;
20 }
21
22 /* THIS DOES NOT WORK */
23
24 int A[3];
25 A = {1, 2, 3};
```



## Arrays are **passed as pointers**

```
4 void print_array(int A[], int n) {  
5     for (int i = 0; i < n; i++) {  
6         printf("%d ", A[i]);  
7     }  
8 }
```

- When you pass an array to a function like this, you're passing a **pointer to the first element**
- Changes to A[i] inside the function **affect** the original array

There is **no built-in .shape or .length** property like in Python.

BE CAREFUL WITH THE **sizeof()** FUNCTION!

- If the array is defined **inside a function** e.g. as `int A[5]`, then **sizeof(A)** does return `5 * sizeof(int)`
- If you want the number of elements in an array, you can use

```
int count = sizeof(a) / sizeof(a[0])
```



divide by the size of int

- This works **only** if the array is in scope as a full array (i.e. was defined inside the function), otherwise it will return the size of the pointer



## Workshop Exercises (Ed)

## Selection Sort

Array:

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

- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort

Array:

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

- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

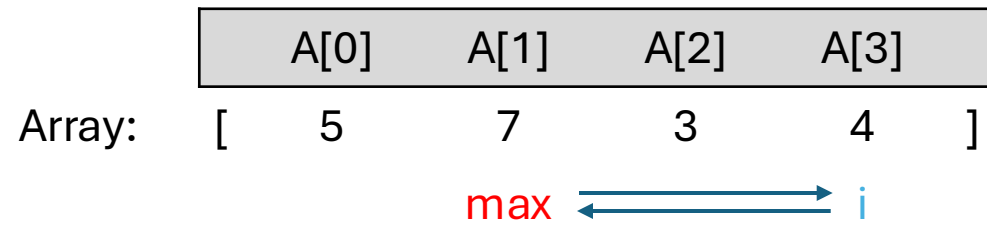
## Selection Sort

Array:

A[0]	A[1]	A[2]	A[3]
[ 5	7	3	4 ]
	max		i

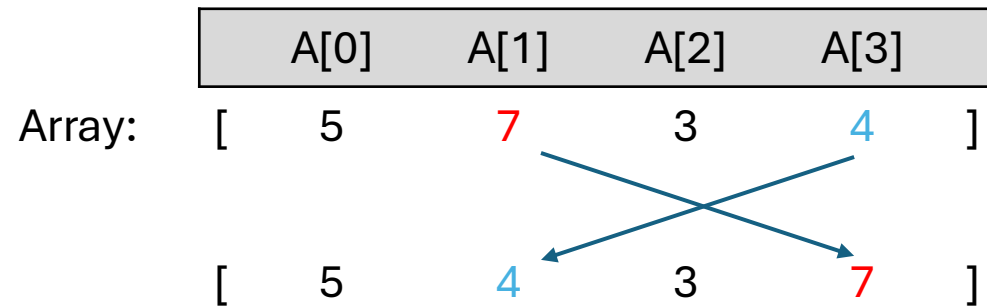
- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort



- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort



- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort

Array:

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

unsorted      sorted

- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort

Array:

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

*i* sorted

- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section



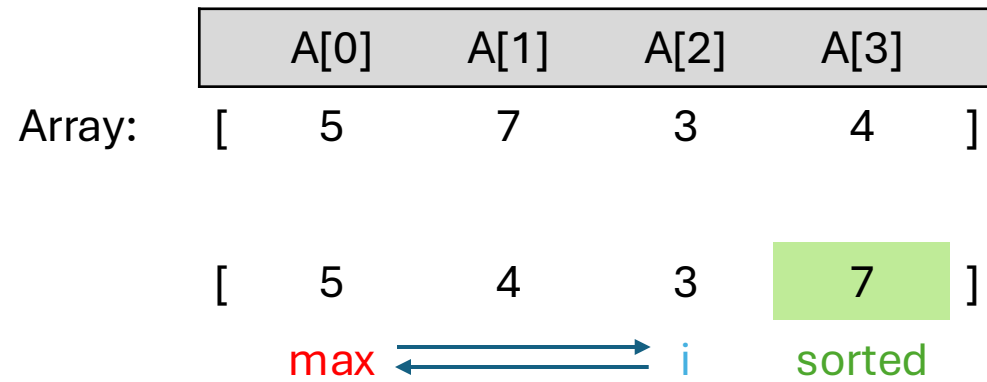
## Selection Sort

Array:

A[0]	A[1]	A[2]	A[3]
5	7	3	4
5	4	3	7
max		i	sorted

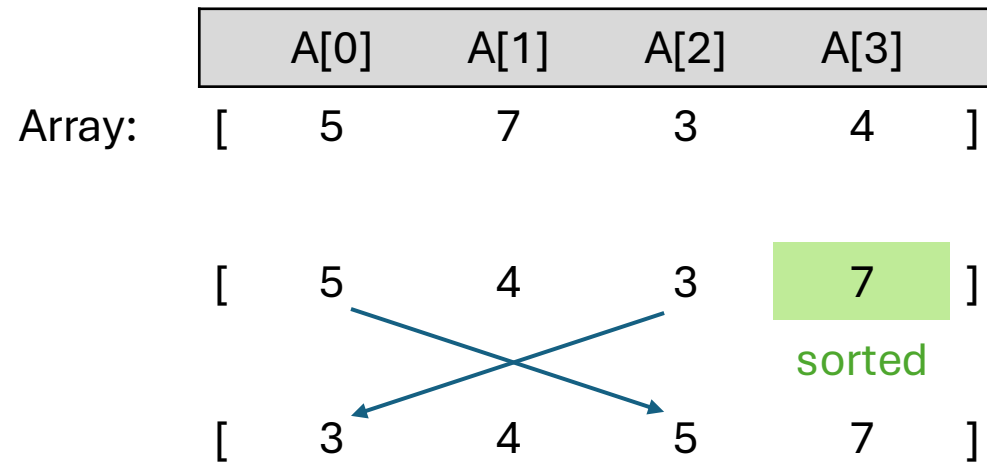
- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort



- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort



- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort

Array:

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

sorted

sorted

- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort

Array:

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

*i* sorted

- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort

Array:

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

*i*  
max

sorted

sorted

- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort

Array:

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

sorted

sorted

- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

## Selection Sort

Array:

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

sorted

sorted

- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section