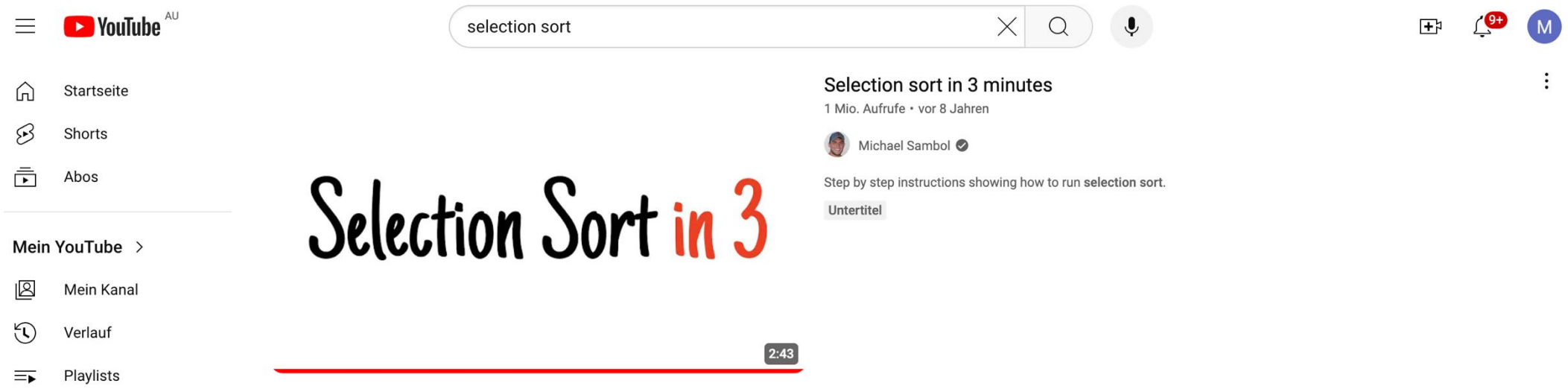
Hello Class!

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



Time Complexity

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

<u>Example 1:</u> 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**

<u>Example 1:</u> 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"

<u>Example 2:</u> 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**

<u>Example 1:</u> 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"

<u>Example 2:</u> 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

	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

```
/*  Initialise directy */

int A[3] = {1, 2, 3};

int A[5] = {1, 2};  // A[2] ... A[4] automatically set to 0

int A[2][3] = {     // 2d array
    {1, 2, 3},
    {4, 5, 6}
}


/* Fill up with a loop */

int A[3];

for (int i = 0; i < 3; i++) {
    A[i] = i + 1;
}

/* THIS DOES NOT WORK */

int A[3];
A = {1, 2, 3};
```

Arrays are **passed as pointers**

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

- 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 Excercises (Ed)

Selection Sort

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

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

Selection Sort

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

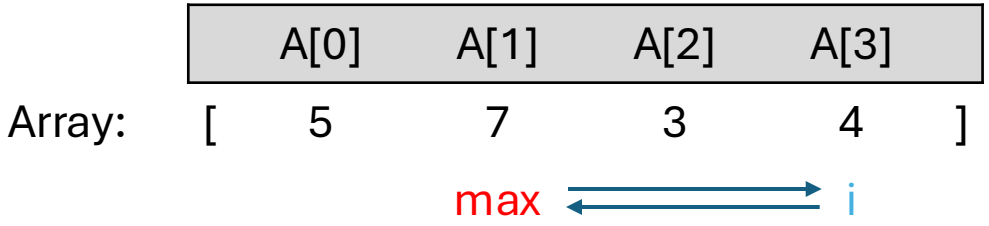
- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

Selection Sort

		A[0]	A[1]	A[2]	A[3]	
Array:	[	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

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

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

Selection Sort

		A[0]	A[1]	A[2]	A[3]	
Array:	[	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

		A[0]	A[1]	A[2]	A[3]	
Array:	[	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

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

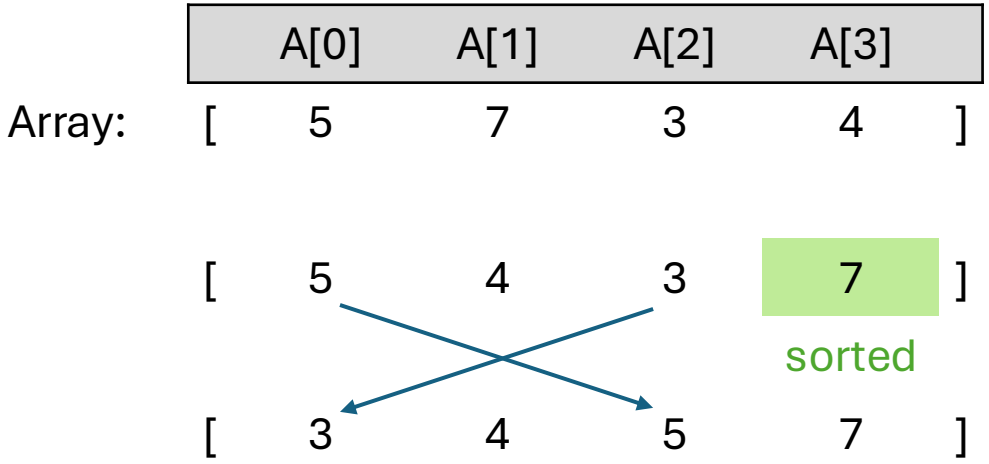
- go through unsorted section of the array
- find the largest value
- swap this value into the sorted section

Selection Sort

		A[0]	A[1]	A[2]	A[3]	
Array:	[	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

		A[0]	A[1]	A[2]	A[3]	
Array:	[	5	7	3	4	]
	[	5	4	3	7	]
					sorted	
	[	3	4	5	7	]
				sorted		

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

Selection Sort

		A[0]	A[1]	A[2]	A[3]	
Array:	[	5	7	3	4	]
	[	5	4	3	7	]
					sorted	
	[	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

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

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

Selection Sort

		A[0]	A[1]	A[2]	A[3]	
Array:	[	5	7	3	4	]
	[	5	4	3	7	]
					sorted	
	[	3	4	5	7	]
				sorted		

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

Selection Sort

		A[0]	A[1]	A[2]	A[3]	
Array:	[	5	7	3	4	]
	[	5	4	3	7	]
					sorted	
	[	3	4	5	7	]
			sorted			

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