TODO BEFORE

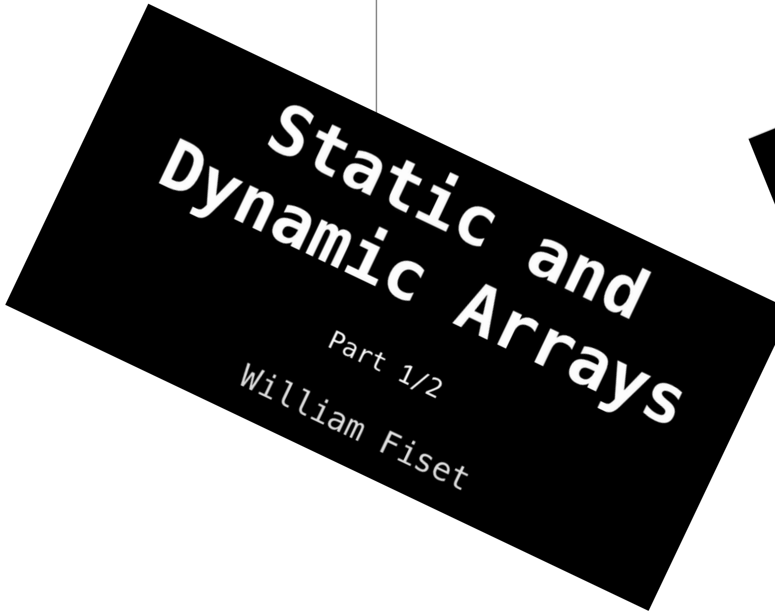
Useful Resource #BeReady

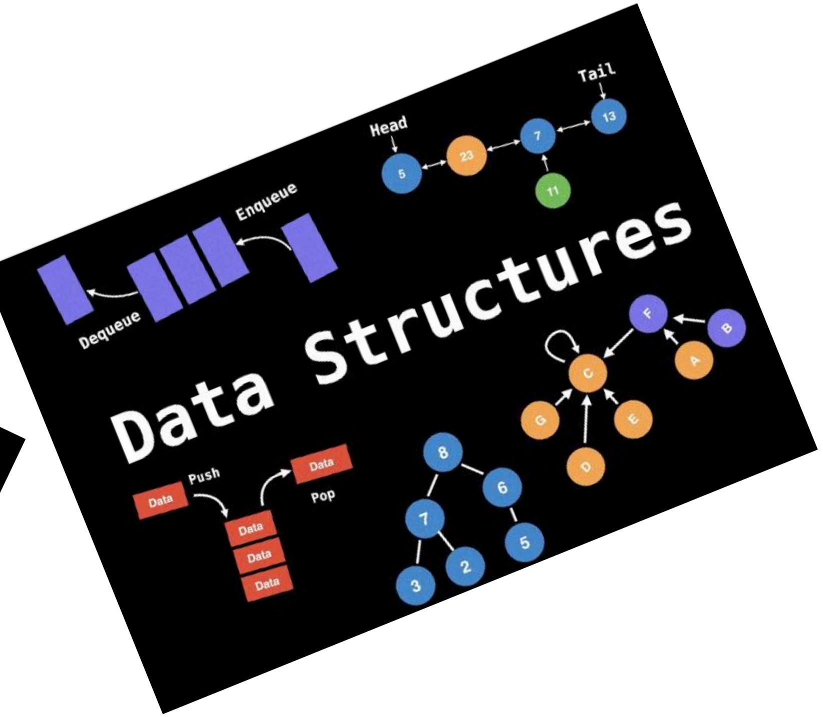
BEFORE NEXT SESSION

- ✓ Watch this first video about ADTs
- ✓ Also watch this video

ALONG THE COURSE !

- ✓ Follow this playlist to understand the most important data structures





ADVANCED ALGORITHM

W4-S1 – Abstract Data Types

🥇 Objectives for today 🥇

- Understand the Concept of **ADTs**
- Differentiate Between **ADTs** and **Data Structures**
- Define the operations of a **Partially Filled Array** and their **complexity**

EXPLAIN

Data Structures

+

Algorithms

=

Efficient Programs

EXPLAIN

How do you **find a word** in a dictionary?

1 - DATA STRUCTURE

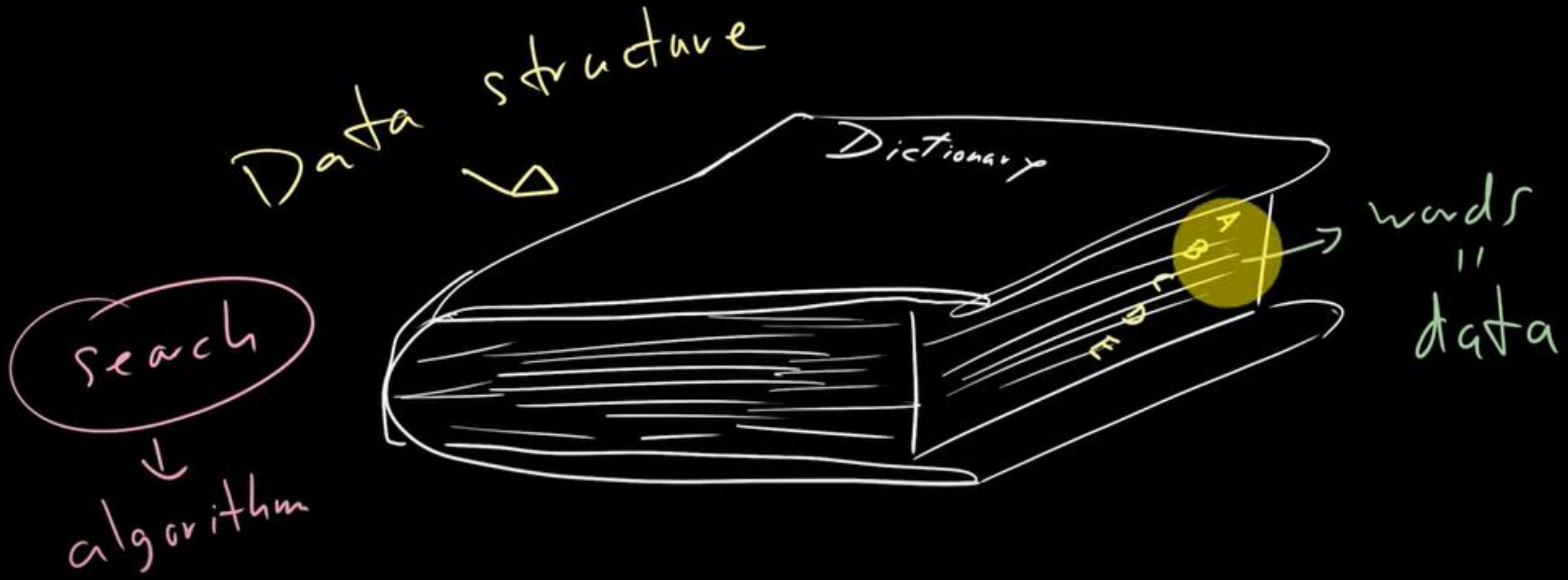
- Words sorted in alphanumerical order

- Indexes

2 - ALGORITHM

- Go to the first letter using indexes

- Then search the word in the sorted list



EXPLAIN

Data Structure (DS)

A data structure is a **format** to help **organize, manage** and **store** data in your program so it can be **accessed** and **modified efficiently.**

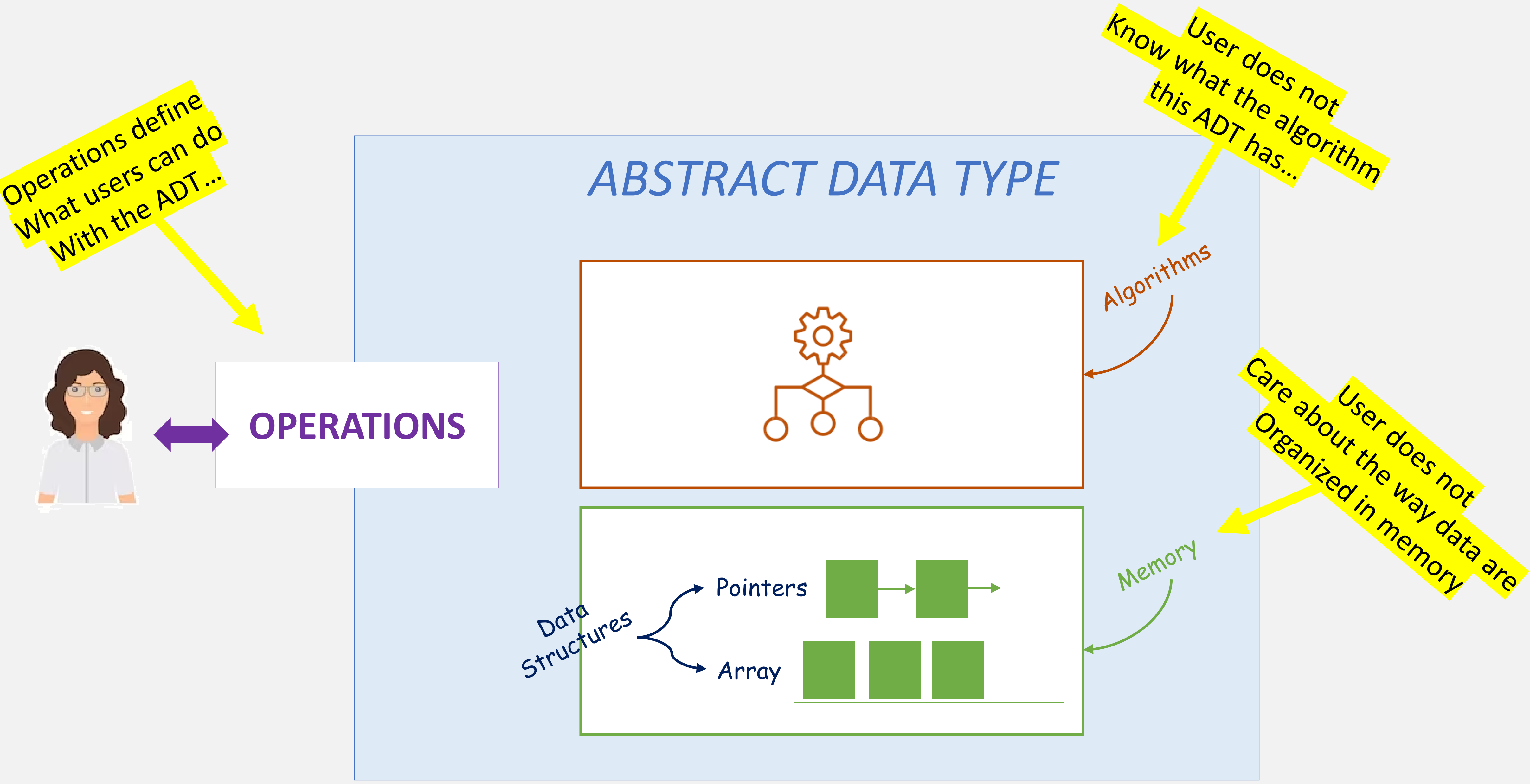
EXPLAIN

Abstract Data Type (ADT)

An Abstract Data Type (ADT) is an abstraction of a data structure which **provides only the interface** to which a data structure must adhere to.

without specific details about how it will be implemented !

ABSTRACT DATA TYPE

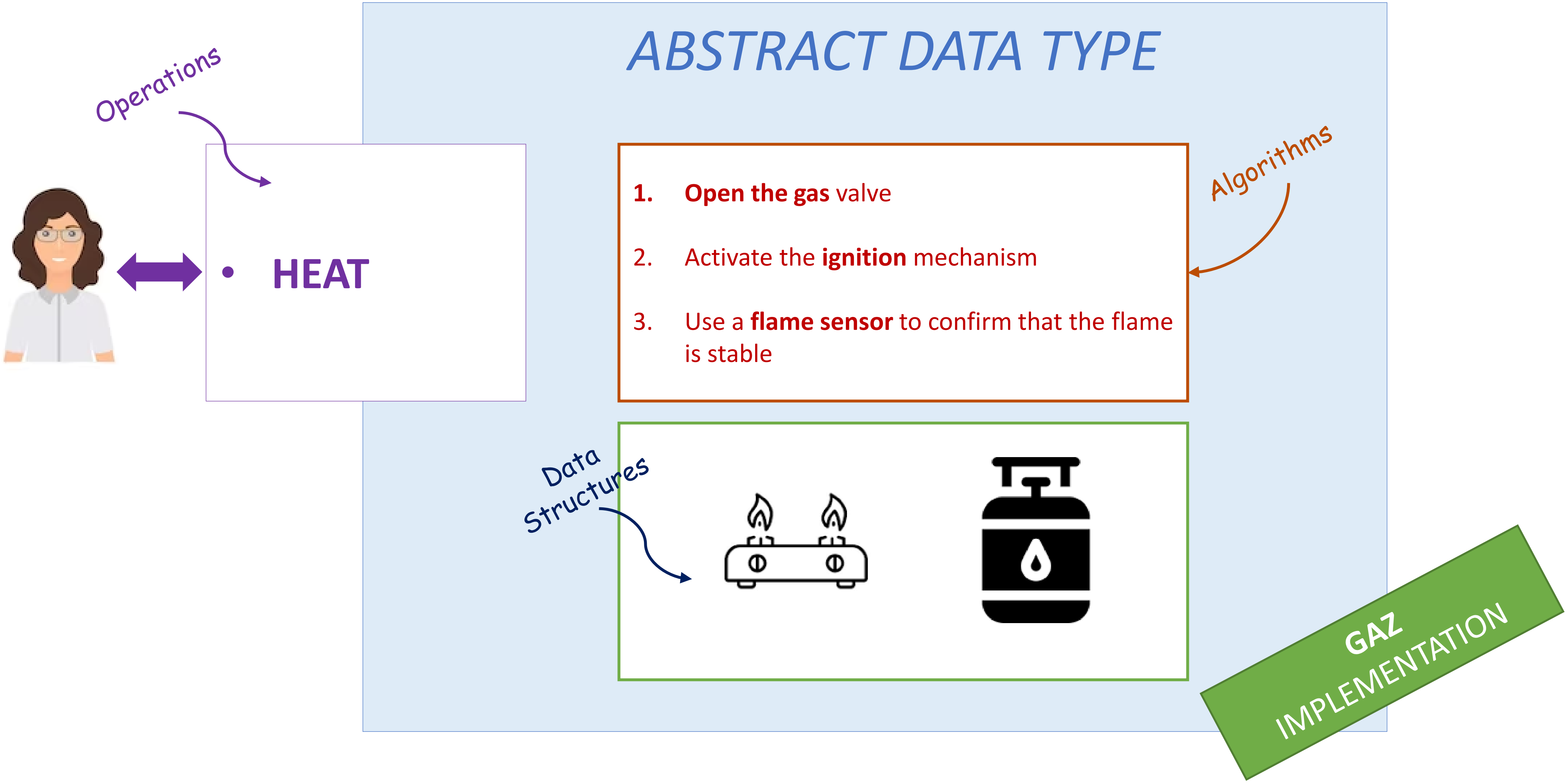


Operations define
What users can do
With the ADT…

User does not
Know what the algorithm
this ADT has…

User does not
Care about the way data are
Organized in memory

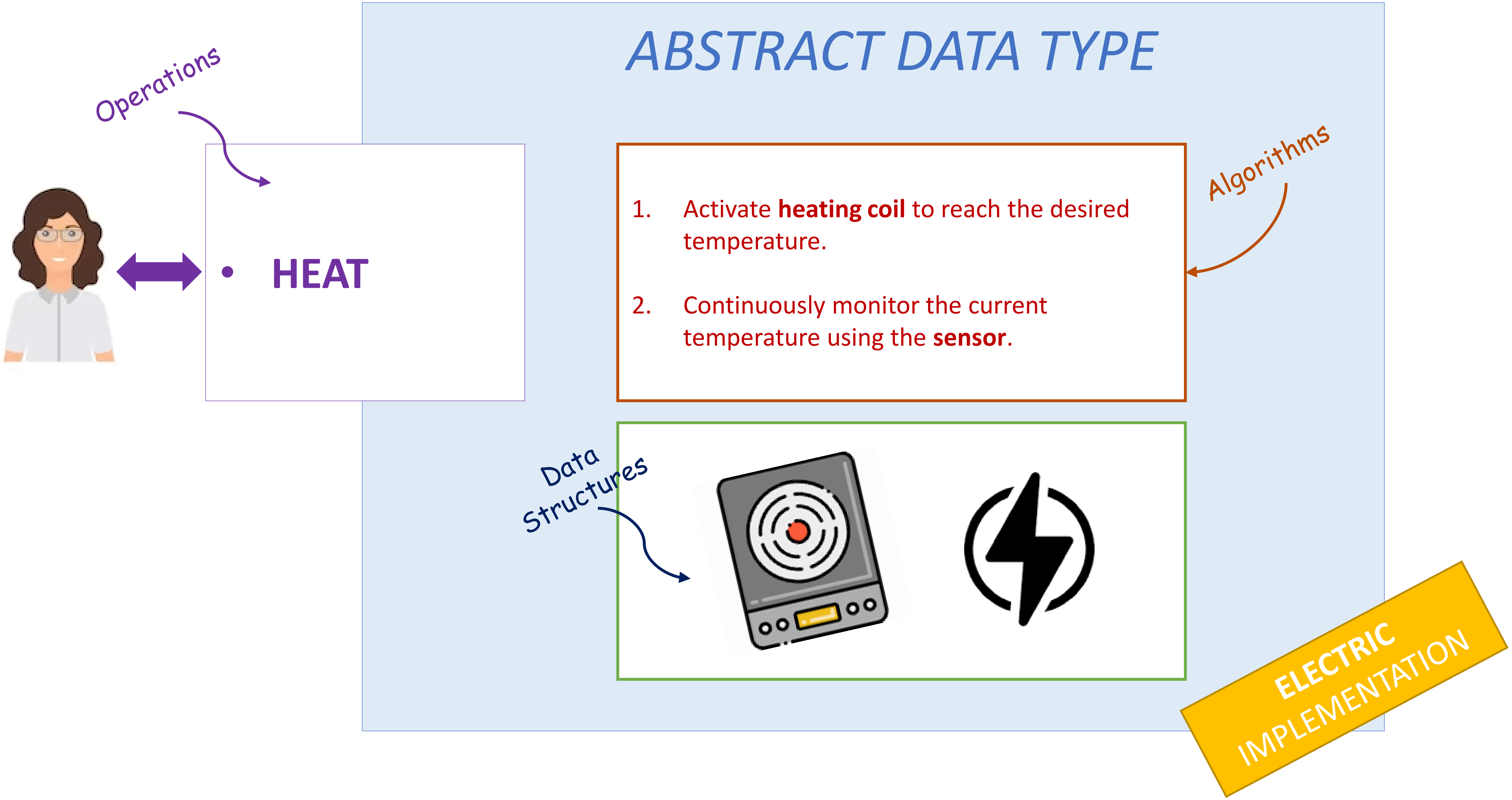
A **cooker** ADT…



ABSTRACT DATA TYPE

1. **Open the gas** valve
2. Activate the **ignition** mechanism
3. Use a **flame sensor** to confirm that the flame is stable

A **cooker** ADT…



ABSTRACT DATA TYPE

Operations

- **HEAT**

Algorithms

1. Activate **heating coil** to reach the desired temperature.
2. Continuously monitor the current temperature using the **sensor**.

Data Structures

ELECTRIC
IMPLEMENTATION

ADT vs Data Structures

Abstraction (ADT)	Implementation (DS)
• List	• Dynamic Array • Linked List
• Stack	• *Array based* stack • *Linked List based* stack
• Vehicle	• Car • Bicycle • Tuk-Tuk

The concept

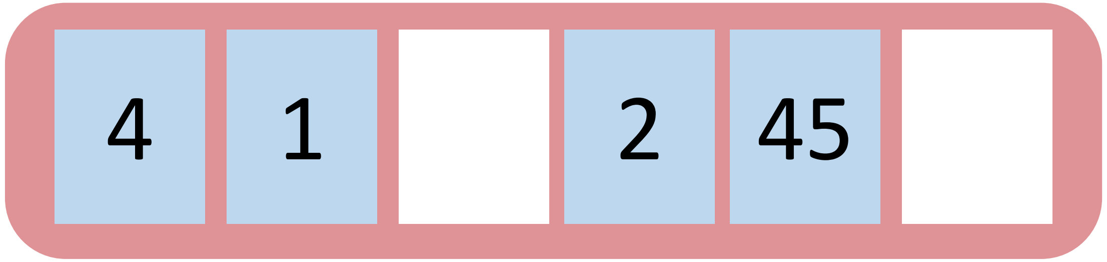
The implementation

EXPLAIN

Partially Filled Array ADT

A partially filled array ADT is an array where **only left part of its capacity holds data**

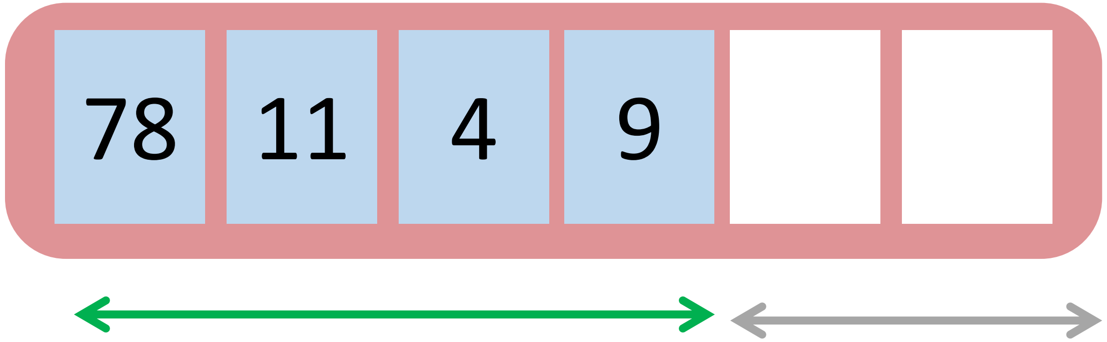
NORMAL ARRAY



4	1		2	45	

No storage rule

PARTIALLY FILLED ARRAY



78	11	4	9		

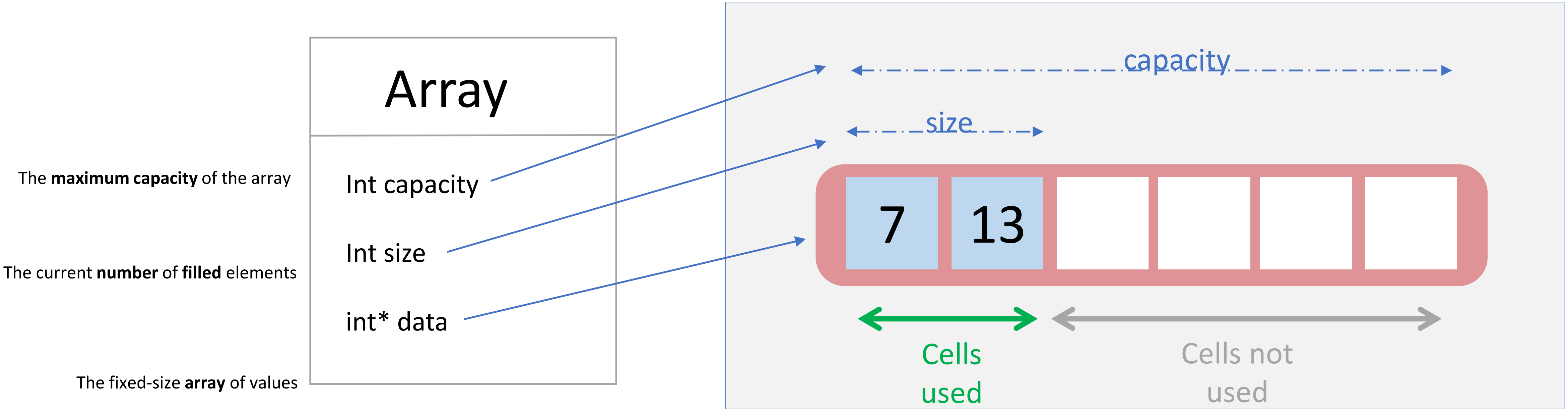
Cells used

Cells not used

EXPLAIN

Partially Filled Array ADT

We can implement a partially filled array using a **class**



Array

Int capacity

Int size

int* data

The **maximum capacity** of the array

The current **number** of **filled** elements

The fixed-size **array** of values

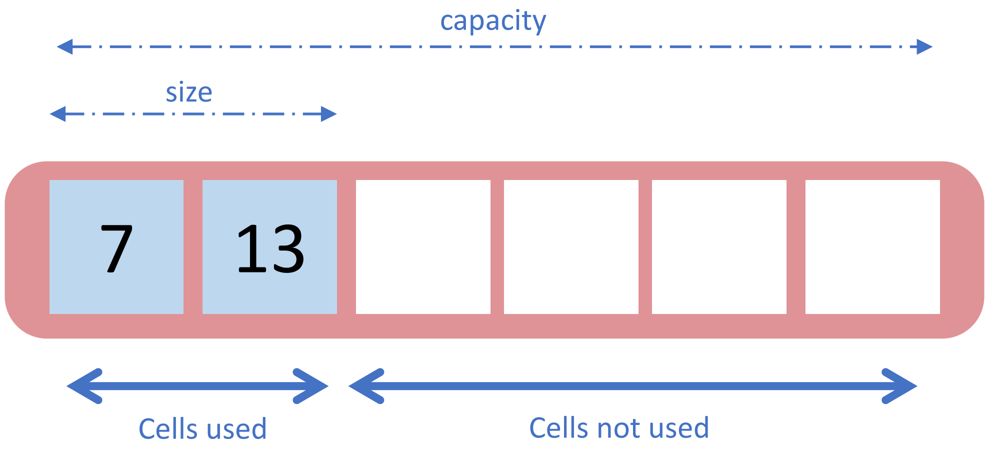
Partially Filled Array ADT

Let’s describe **the specifications** of 3 operations on this ADT

OPERATIONS

1. **int get(index)**
2. **void insertAtEnd(value)**
3. **void insertAtStart(value)**

DATA STRUCTURE

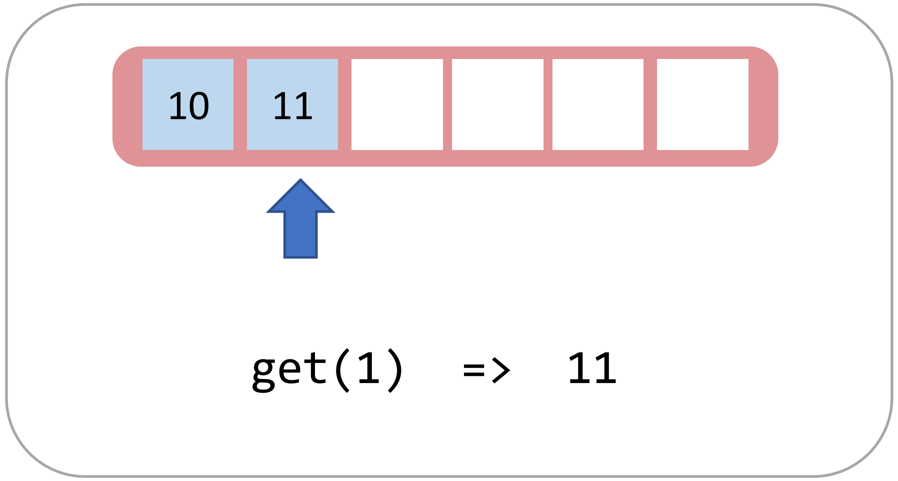


1 int get(index)

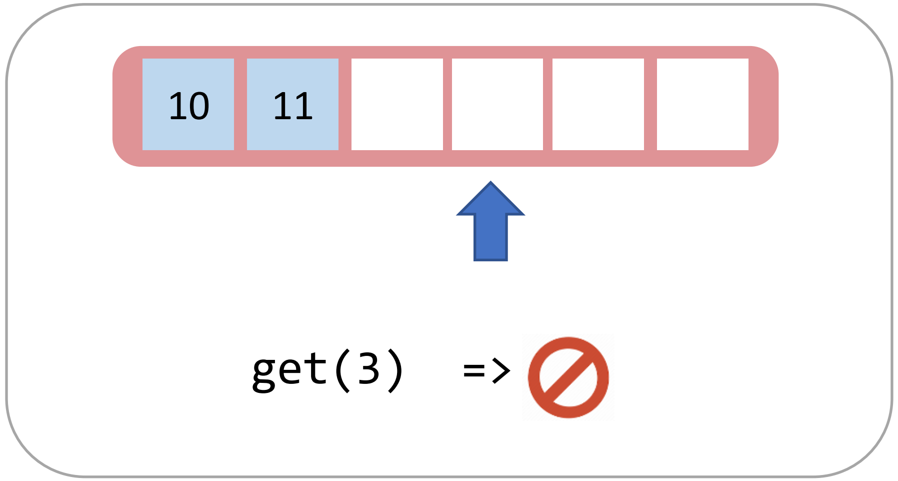
Syntax	int get(int index)
Description	Get the **value of the cell** at given index
Precondition	The index must be in the range 0… size-1

Example	`myList = [10, 11]`
	`int value = myList.get(1)`
	-> value should be 11

Complexity	O(1) Only 1 operation needed to access to the value



CASE 1



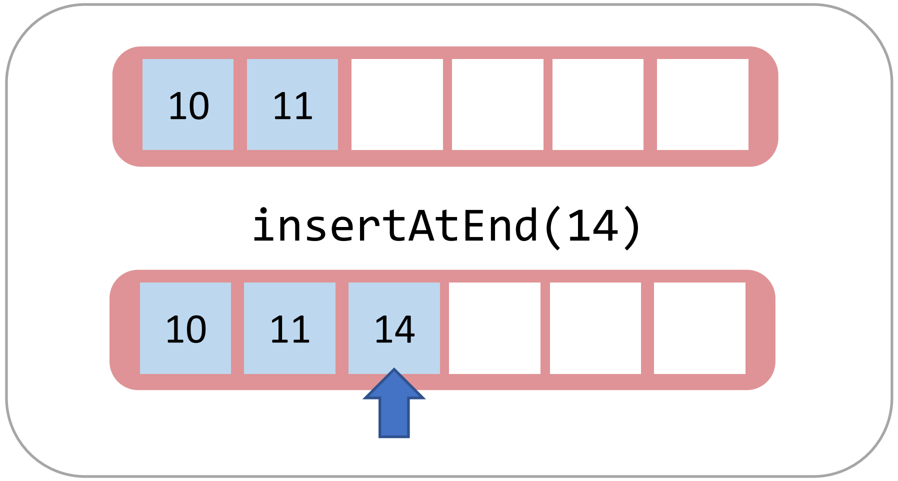
CASE 2

2 void insertAtEnd(value)

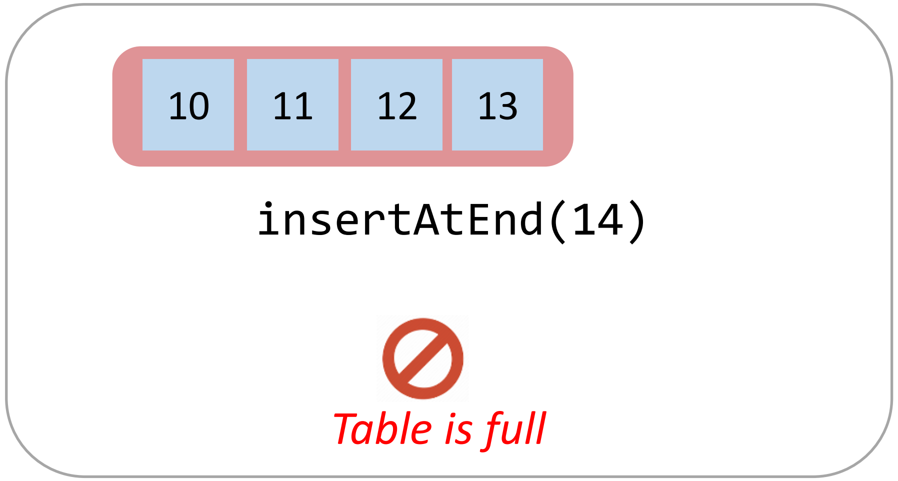
Syntax	int insertAtEnd(int index)
Description	Insert a **value at the end** of the array
Precondition	Capacity-size > 0

Example	`myList = [10, 11]`
	`myList.insertAtEnd(14)`
	
	-> myList is now [10, 11,14]

Complexity	O(1) Only 1 operation needed to insert at the end



CASE 1



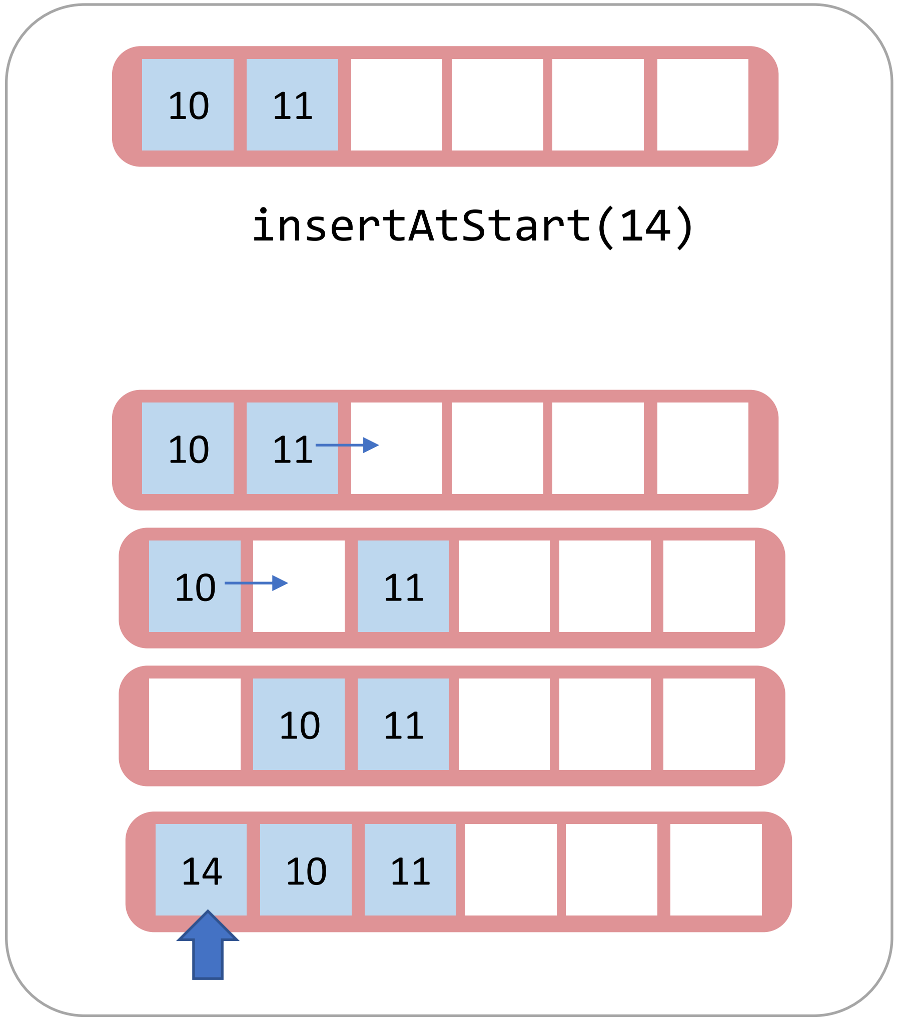
CASE 2

3 void insertAtStart(value)

Syntax	int insertAtStart(int index)
Description	Insert a **value at the begining** of the array
Precondition	Capacity-size > 0

Example	`myList = [10, 11]`
	`myList.insertAtStart(14)`
	-> myList is now [14, 10, 11]

Complexity	O(n) We need N operations (moving N elements to the right



ACTIVITY

10 MIN

Insert !!

The operation **insertAt** *inserts a value at a given index in the array*

Q1 - Define the **specifications** of this operation

Syntax	
Description	
Precondition	
Example	
Complexity	

Q2 – Identify different use cases

```
[10,11,--,--]
insertAt(1,12)
[10,12,11,--]
```

```
[10,11,12]
insertAt(1,13)
Array is full
```

CORRECTION

10 MIN

Insert !!

The operation **insertAt** *inserts a value at a given index in the array*

Q1 - Define the **specifications** of this operation

Syntax	int insertAt(int index, int value)
Description	Insert a **value at given index in** the array
Precondition	index in range 0… size capacity-size > 0

Example	`myList = [10, 11]`
	`myList.insertAt(1,14)`
	-> myList is now [10, 14, 11]

Complexity	O(n) We consider the worst case !! *(every elements need to be switched)*

Q2 – Identify different use cases

```
[10,11,--,--]
insertAt(1,12)
[10,12,11,--]
```

```
[10,11,--,--]
insertAt(2,12)
[10,11,12,--]
```

```
[10,11,12]
insertAt(1,13)
Array is full
```

```
[10,11,--]
insertAt(3,13)
Wrong index
```

```
[10,11,--]
insertAt(-1,13)
Wrong index
```

ACTIVITY

10 MIN

Search !!

The operation ***search*****(value)** *search for an index with a given value in the array*

Q1 - Define the **specifications** of this operation

Syntax	
Description	
Precondition	
Example	
Complexity	

Q2 – Identify different use cases

```
[10,11,12,13]
search(11)
=>1
```

```
[10,11,12,13]
Get(4)
=> Not found
```

```
[10,11,12,13]
search(13)
=>3
```

CORRECTION

10 MIN

Activity !!

The operation ***search*****(value)** *search for an index with a given value in the array*

Q1 - Define the **specifications** of this operation

Syntax	int search(int value)
Description	*search for a given value in the array*
Precondition	size > 0

Example	`myList= [10,11,12,13]`
	`myList.search(11)`
	=>1

Complexity	O(n) We consider the worst case !!

Q2 – Identify different use cases

```
[10,11,12,13]
search(11)
=>1
```

```
[10,11,12,13]
search(4)
=> Not found
```

```
[10,11,12,13]
search(13)
=>3
```

```
[10,13,12,13]
search(13)
=>1
```

ACTIVITY

10 MIN

Activity !!

The operation **removeAt()** *remove a value at a given index in the array*

Q1 - Define the **specifications** of this operation

Syntax	
Description	
Precondition	
Example	
Complexity	

Q2 – Identify different use cases

```
[10,11,12,13]
removeAt(1)
[10,12,13,--]
```

```
[10,12,13,--]
removeAt(1)
[10,13,--,--]
```

```
[--,--,--]
removeAt(1)
Array is empty
```

```
[10,11,--]
removeAt(3)
Wrong index
```

```
[10,11,--]
removeAt(-1)
Wrong index
```

CORRECTION

10 MIN

Activity !!

The operation **removeAt()** *remove a value at a given index in the array*

Q1 - Define the **specifications** of this operation

Syntax	int removeAt(int index)
Description	remove a **value at given index in** the array
Precondition	index >= 0 Index < size

Example	`myList = [10,11,12,13]`
	`myList = removeAt(1)`
	
	-> myList is now [10, 12, 13]

Complexity	O(n) We consider the worst case !! *(The elements need to be shift)*

Q2 – Identify different use cases

```
[10,11,12,13]
removeAt(1)
[10,12,13,--]
```

```
[10,12,13,--]
removeAt(1)
[10,13,--,--]
```

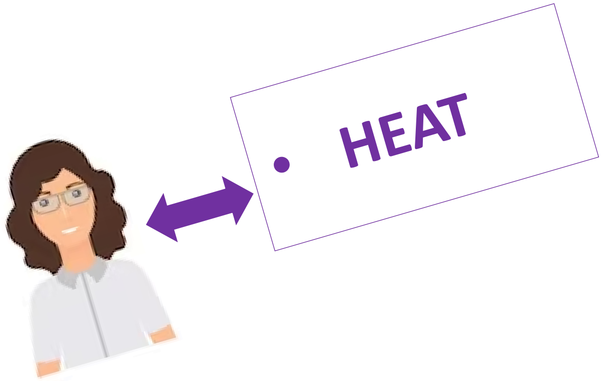
```
[--,--,--]
removeAt(1)
Array is empty
```

```
[10,11,--]
removeAt(3)
Wrong index
```

```
[10,11,--]
removeAt(-1)
Wrong index
```

In Short

When designing an ADT we care about:

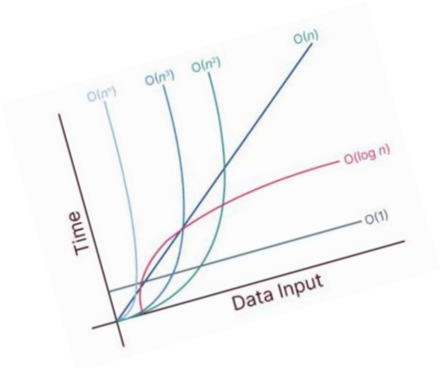


- HEAT

✓ The **operations** (*interface*) of this ADT

✓ The **implementation**
- Data Structure
- Algorithm

✓ The **cost**
- Time
- Memory



Resources – *Sorting Algorithms*

To go further…

Follow this playlist to understand the most important data structures

🥇 You should know… 🥇

✓ Understand the Concept of **ADTs**

✓ Differentiate Between **ADTs** and **Data Structures**

✓ Define the operations of a **Partially Filled Array** and their **complexity**

WRAPPING THE LESSON

3-2-1 Challenge

- ✓ List three things you **learned** today.
- ✓ List two **questions** you still have.
- ✓ List one aspect of the lesson or topic you **enjoyed**.