Arrays

Data Structures

Thus far, all of our data has been stored in variables

- one variable holds one piece of data

Data structures enable our programs to organize our data in more efficient, sensible ways

- group related pieces of data together

We'll see three types of data structures this semester

- variables (all semester)
- arrays (this week)
- classes (in a few weeks)

Exercise: Storing Multiple Pieces of Data

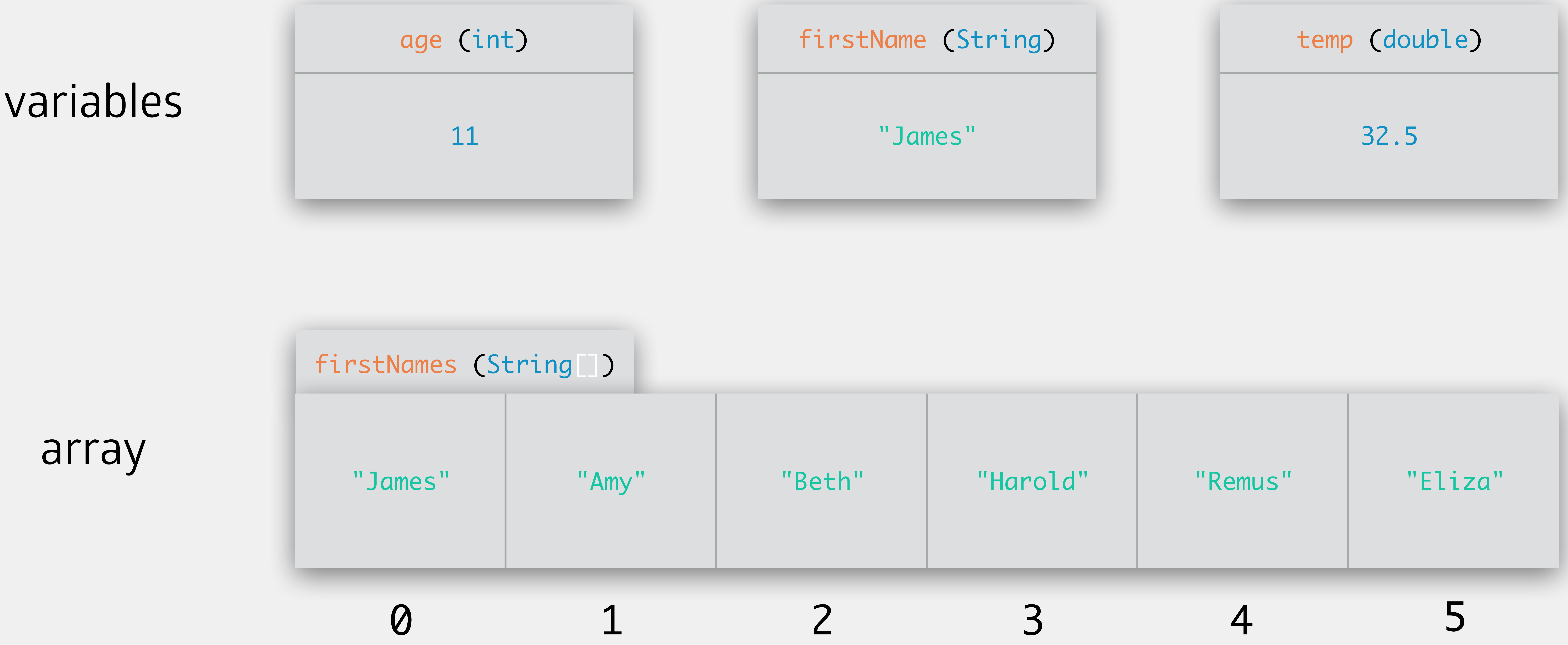
Suppose we wanted to store the names of everyone in this class

What information do we need to know?

How can we store that information in a program?

What if the user was providing the names through the console? Could we adapt to changes to how many people are in the class? (e.g., 27 vs 33?)

What Is An Array?



variables

age (int)

11

firstName (String)

"James"

temp (double)

32.5

array

firstNames (String[])

0	1	2	3	4	5
"James"	"Amy"	"Beth"	"Harold"	"Remus"	"Eliza"

Array Properties

Arrays allow us to store a collection of data values together

All data stored in an array must be of the same data type

- e.g., all Strings, all ints, all booleans

Must predetermine the size of our array

- e.g., if we say our array will hold 27 names, we cannot modify it to store 33 names
- however, we can always store less data (e.g., 15 names)

We refer to data by its variable name **and** index (i.e., position) in the array

- indexes are zero-based, just like with Strings
- the length of the String is **not** zero-based

Setting Up An Array

Three steps:

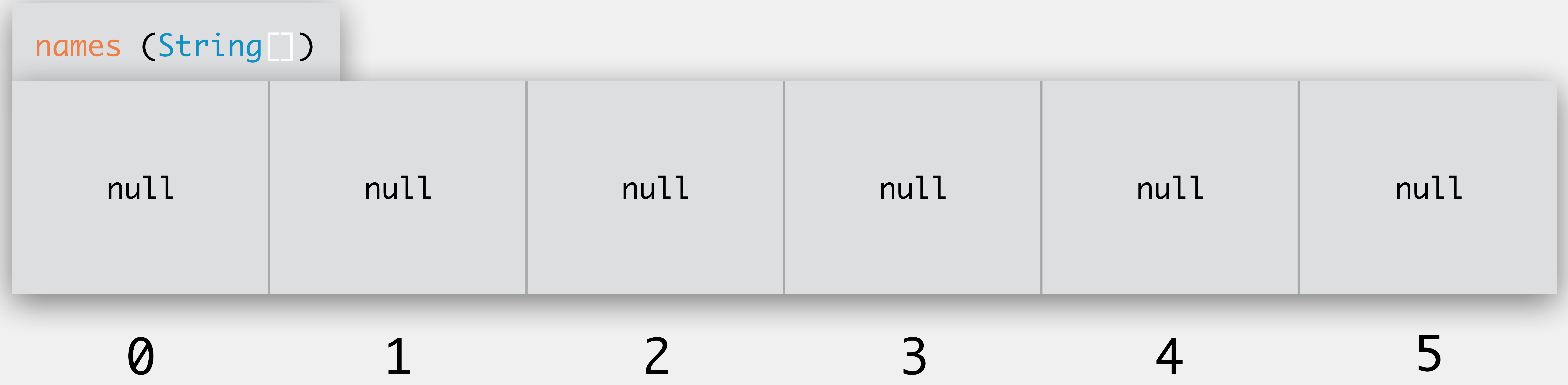
- *declaring* the array sets up the variable name and data type
  - only change is the addition of square brackets, e.g., `[]`

```
names (String[])
```

Setting Up An Array

Three steps:

- *declaring* the array sets up the variable name and data type
  - only change is the addition of square brackets, e.g., `[]`
- *instantiating* the array sets up the size (i.e., length)



`names (String[])`

null	null	null	null	null	null
0	1	2	3	4	5

What Is Null?

The absence of data

Keyword in Java to indicate that there is nothing (i.e., no data) referred to by this variable/spot in the array

Always (always always) initialize/instantiate variables/arrays!

- except for primitives, these are set to null until initialization/instantiation

NullPointerException

Java throws an exception when your program attempts to use a null values

- accessing an array that has not been instantiated
- accessing a spot in the array that has not been initialized
- will see this other places too (e.g., classes)

```
Exception in thread "main" java.lang.NullPointerException
    at Example.main(Example:8)
```

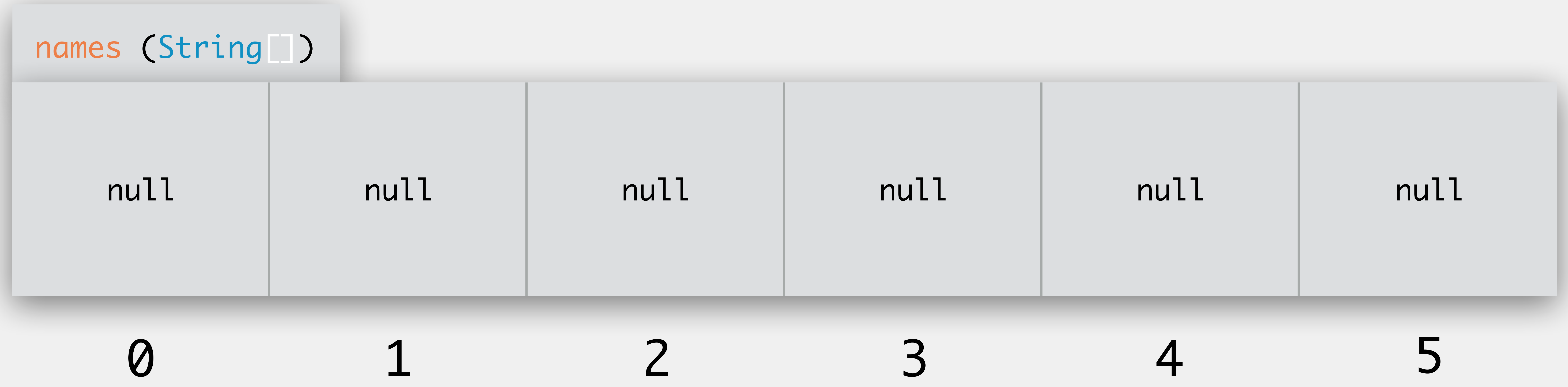
name of the exception that caused our program to crash

line number where the exception occurred

Setting Up An Array

Three steps:

- *declaring* the array sets up the variable name and data type
  - only change is the addition of square brackets, e.g., `[]`
- *instantiating* the array sets up the size (i.e., length)



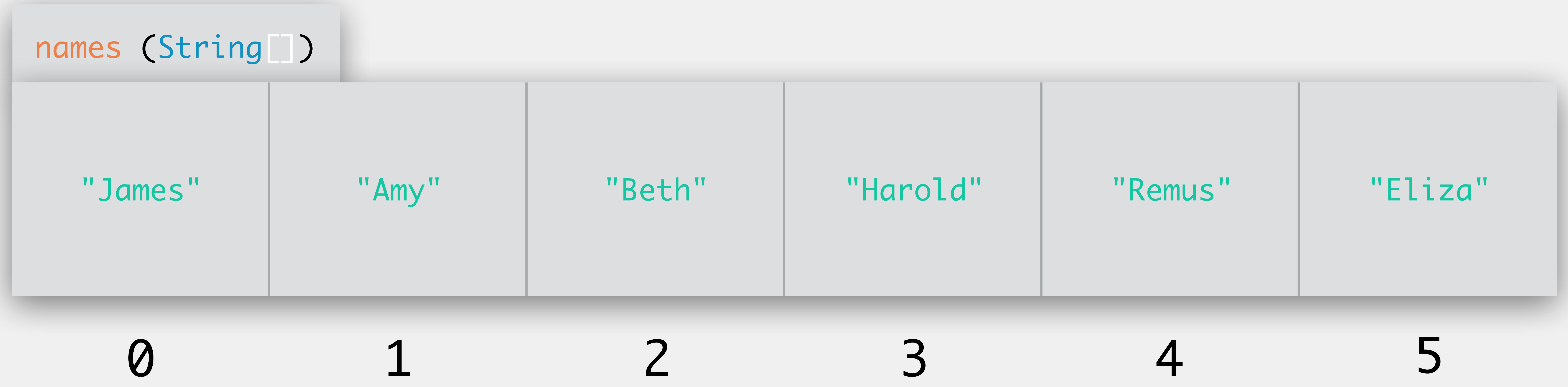
names (String[])

null	null	null	null	null	null
0	1	2	3	4	5

Setting Up An Array

Three steps:

- *declaring* the array sets up the variable name and data type
  - only change is the addition of square brackets, e.g., []
- *instantiating* the array sets up the size (i.e., length)
- *initializing* the array assigns initial values to each spot in the array



names (String[])

"James" "Amy" "Beth" "Harold" "Remus" "Eliza"

0 1 2 3 4 5

Definition: Declaring & Instantiating An Array

declare an array

```
<dataType>[] <identifier>;   // both of these lines do the same thing
<dataType> <identifier>[];
```

instantiate an array

```
<identifier> = new <dataType>[<length>];
```

declare and instantiate an array

```
<dataType>[] <identifier> = new <dataType>[<length>];
```

Example: Declaring & Instantiating An Array

declare an array of type String called names

```
String[] names;    // both of these lines do the same thing
String names[];
```

instantiate an array of type String with length 6

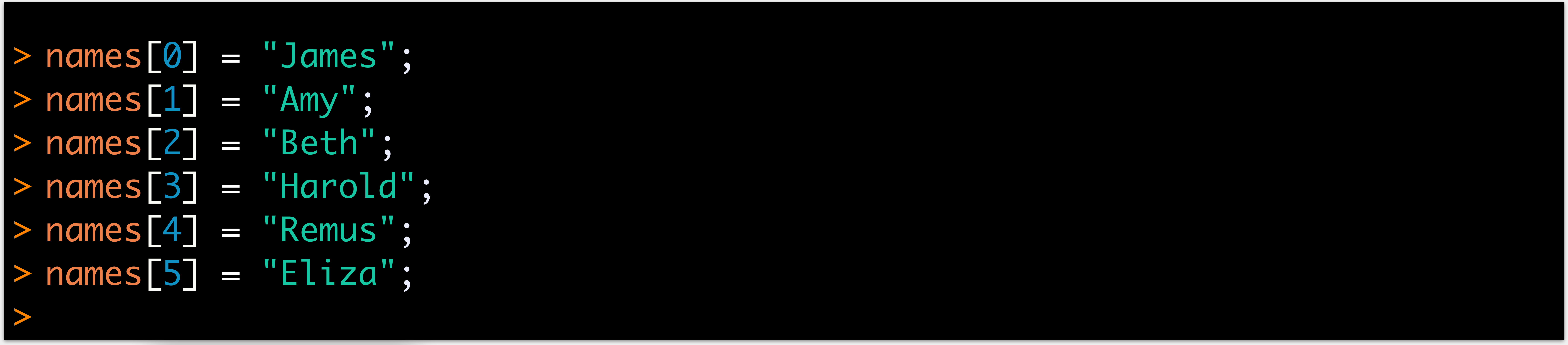
```
names = new String[6];    // notice we do not use the square brackets here
```

declare and instantiate an array of type String called names with length 6

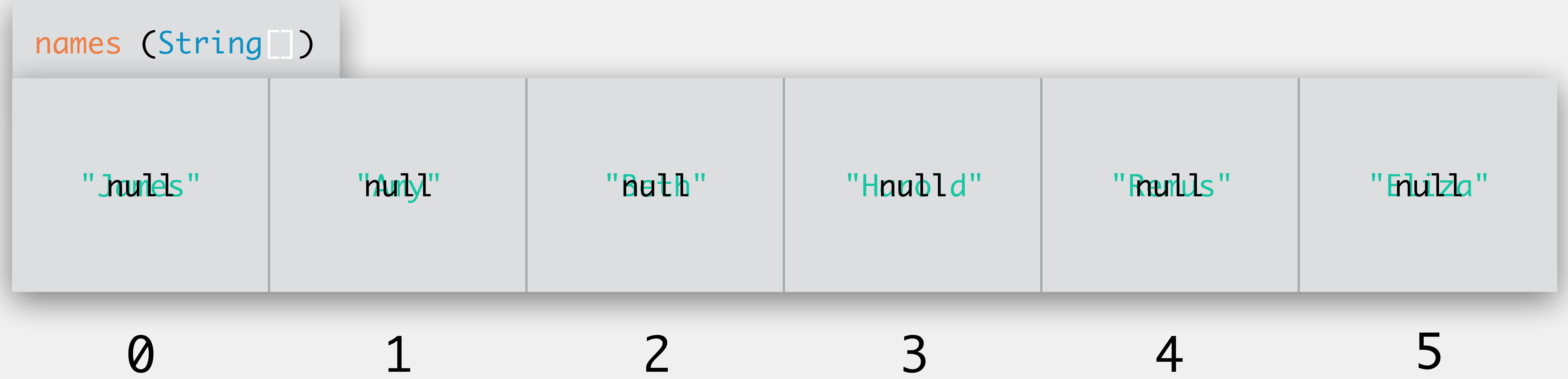
```
String[] names = new String[6];
```

Example: Array Initialization

initialize an array of type String called names



```
> names[0] = "James";
> names[1] = "Amy";
> names[2] = "Beth";
> names[3] = "Harold";
> names[4] = "Remus";
> names[5] = "Eliza";
>
```



Example: Declaring, Instantiating, and Initializing

declare an array of type String called names

```
String[] names;    // both of these lines do the same thing
String names[];
```

instantiate and initialize an array with our name Strings

```
names = {"James", "Amy", "Beth", "Harold", "Remus", "Eliza"};
```

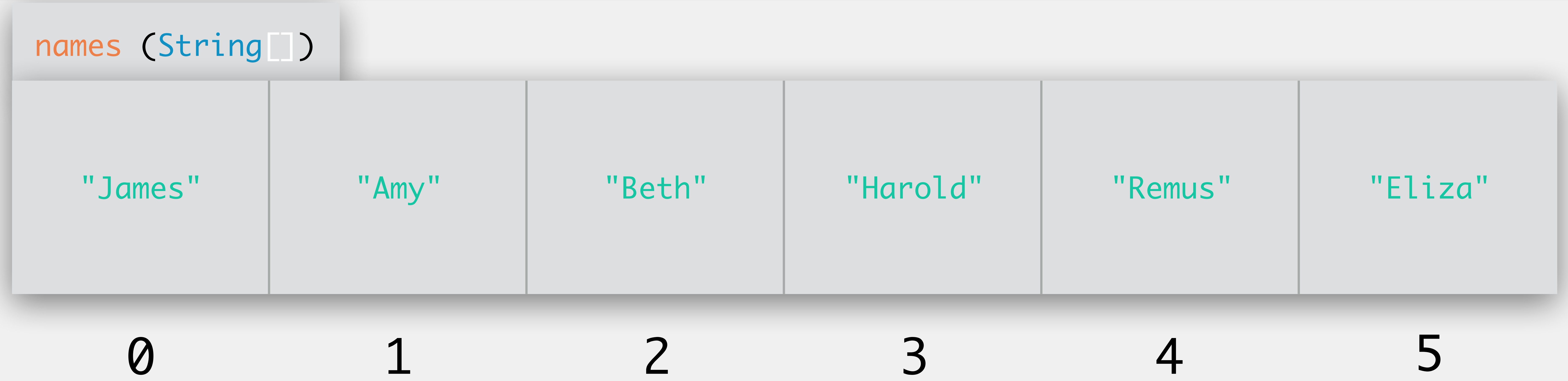
declare, instantiate and initialize an array with our name Strings

```
String[] names = {"James", "Amy", "Beth", "Harold", "Remus", "Eliza"};
```

Example: Array Access

access each value in the array and print it out

```
System.out.println(names[0]);
System.out.println(names[1]);
System.out.println(names[2]);
System.out.println(names[3]);
System.out.println(names[4]);
System.out.println(names[5]);
```



names (String[])

"James"	"Amy"	"Beth"	"Harold"	"Remus"	"Eliza"
0	1	2	3	4	5

Definition: Array Length

Like Strings, can often be helpful to know the length of an array

Unlike Strings, we use `.length`

- notice no parentheses!

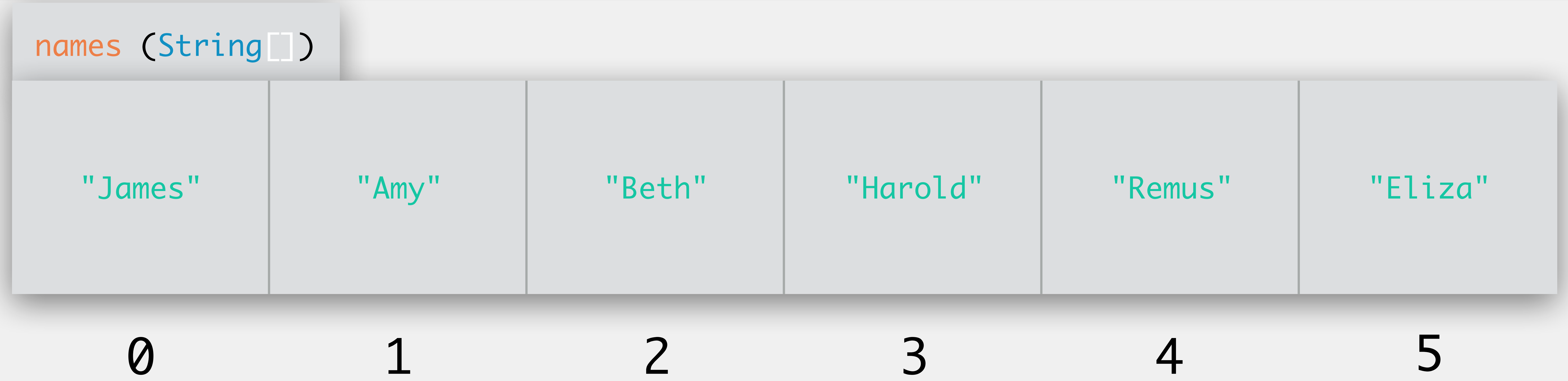
access the length of an array

```
<identifier>.length;
names.length;
```

Example: Array Access

access each value in the array and print it out

```
for (int i = 0; i < names.length; ++i) {
    System.out.println(names[i]);
}
```



names (String[])

0	1	2	3	4	5
"James"	"Amy"	"Beth"	"Harold"	"Remus"	"Eliza"

ArrayIndexOutOfBoundsException

Java throws an exception when your program attempts to access a value beyond the length of the array

- similar to attempting to access a character index not available in a String

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
    at Example.main(Example:8)
```

name of the exception that caused our program to crash

line number where the exception occurred

Definition: `String` Methods

`toCharArray`: converts a `String` to an array of `char` values

```
str.toCharArray();
```

`equals`: checks for equality between one `String` and another (case sensitive!)

```
str.equals(str2);
```

`==`: checks to see if two `String` values point to the same memory location

```
str == str2;
```

toCharArray

arguments: nothing

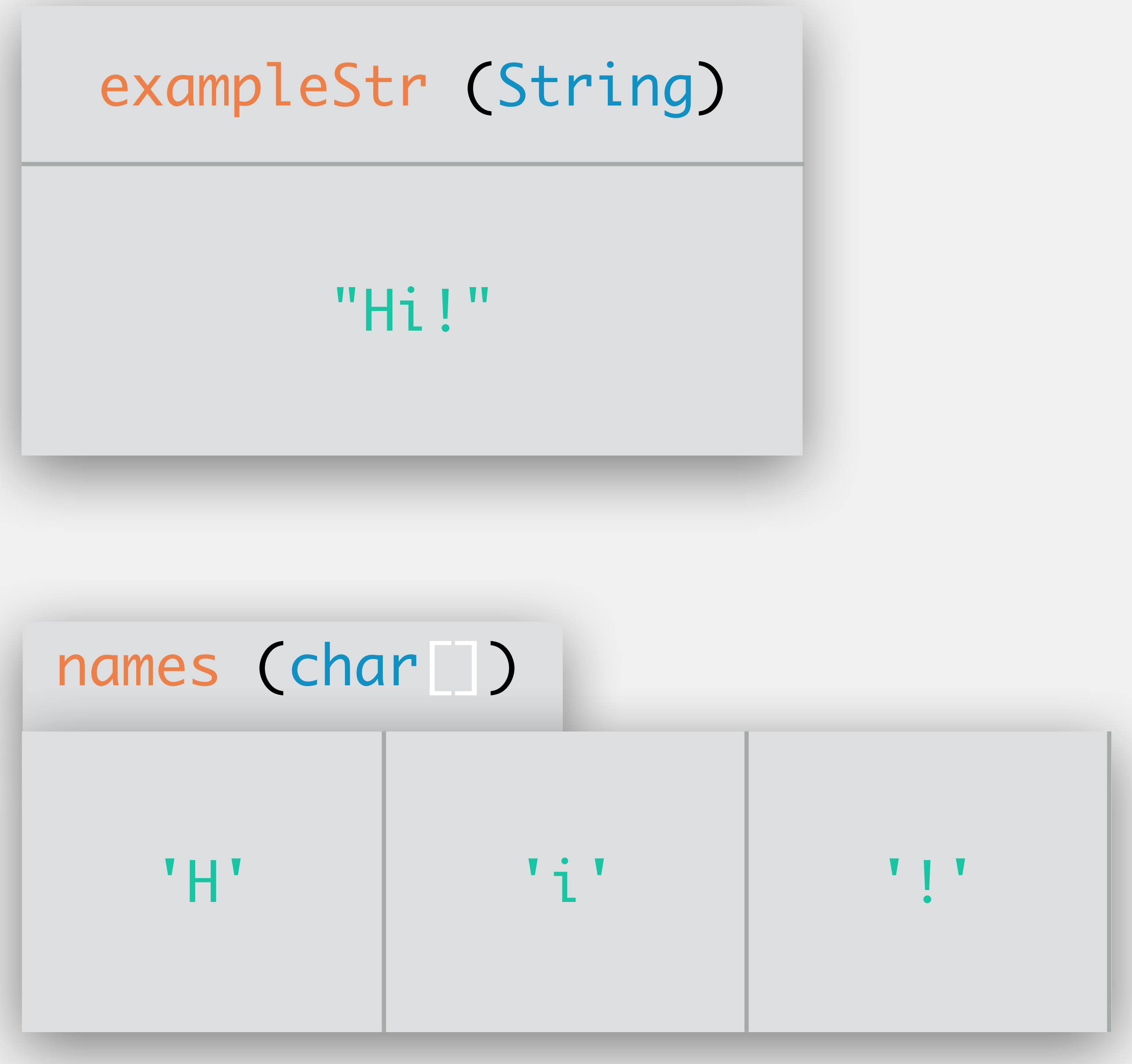
returns: a char array containing each character in the String, in order

```
<String>.toCharArray();
```

```
> String exampleStr = "Hi!";

> char[] arr = exampleStr.toCharArray();
>
```

memory



equals

arguments: a String to compare to

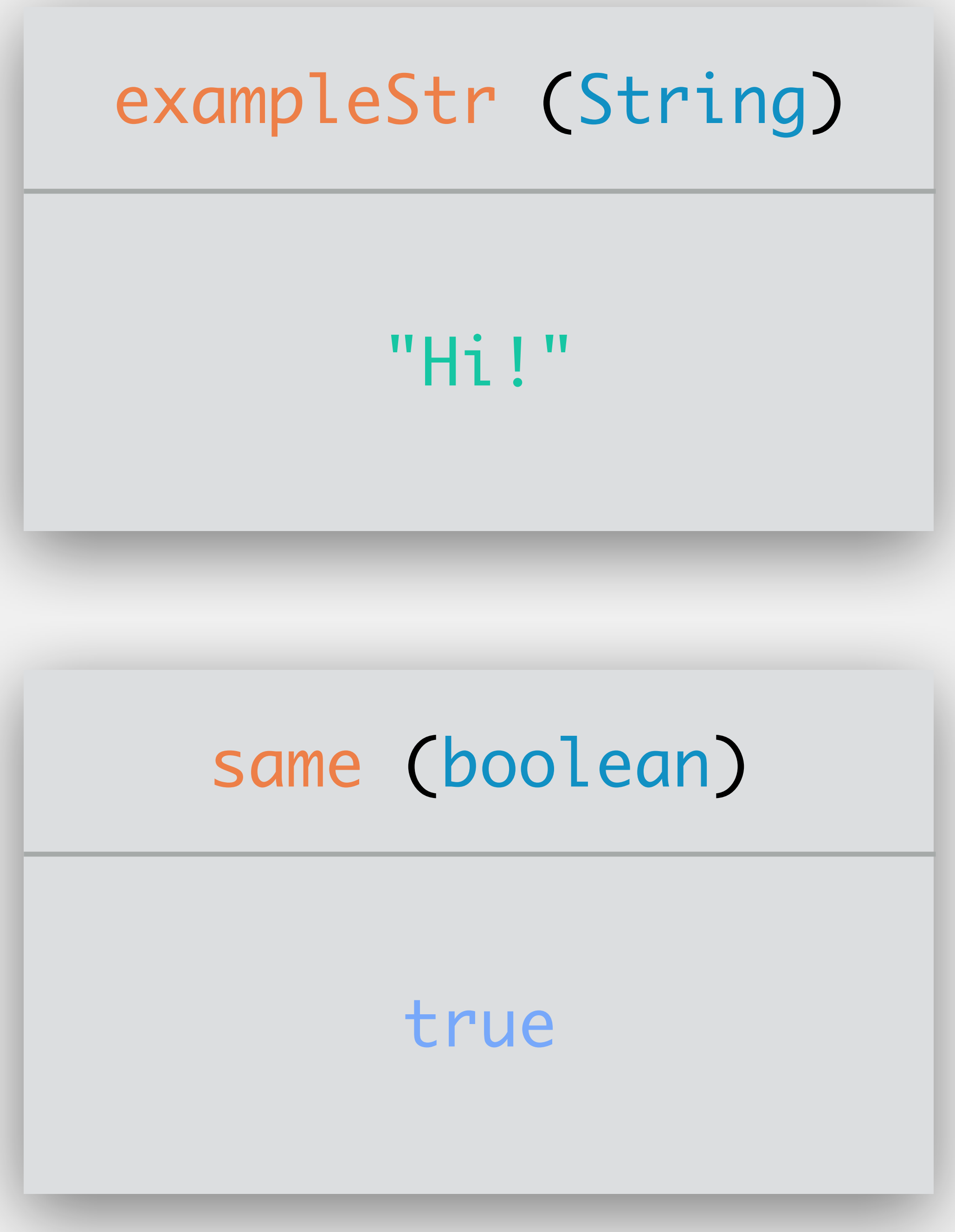
returns: a boolean value; true if the two Strings are the same, false if not

```
<String>.equals(<String>);
```

```
> String exampleStr = "Hi!";

> boolean same = exampleStr.equals("Hi!");
>
```

memory



exampleStr (String)

"Hi!"

same (boolean)

true

equals

arguments: a String to compare to

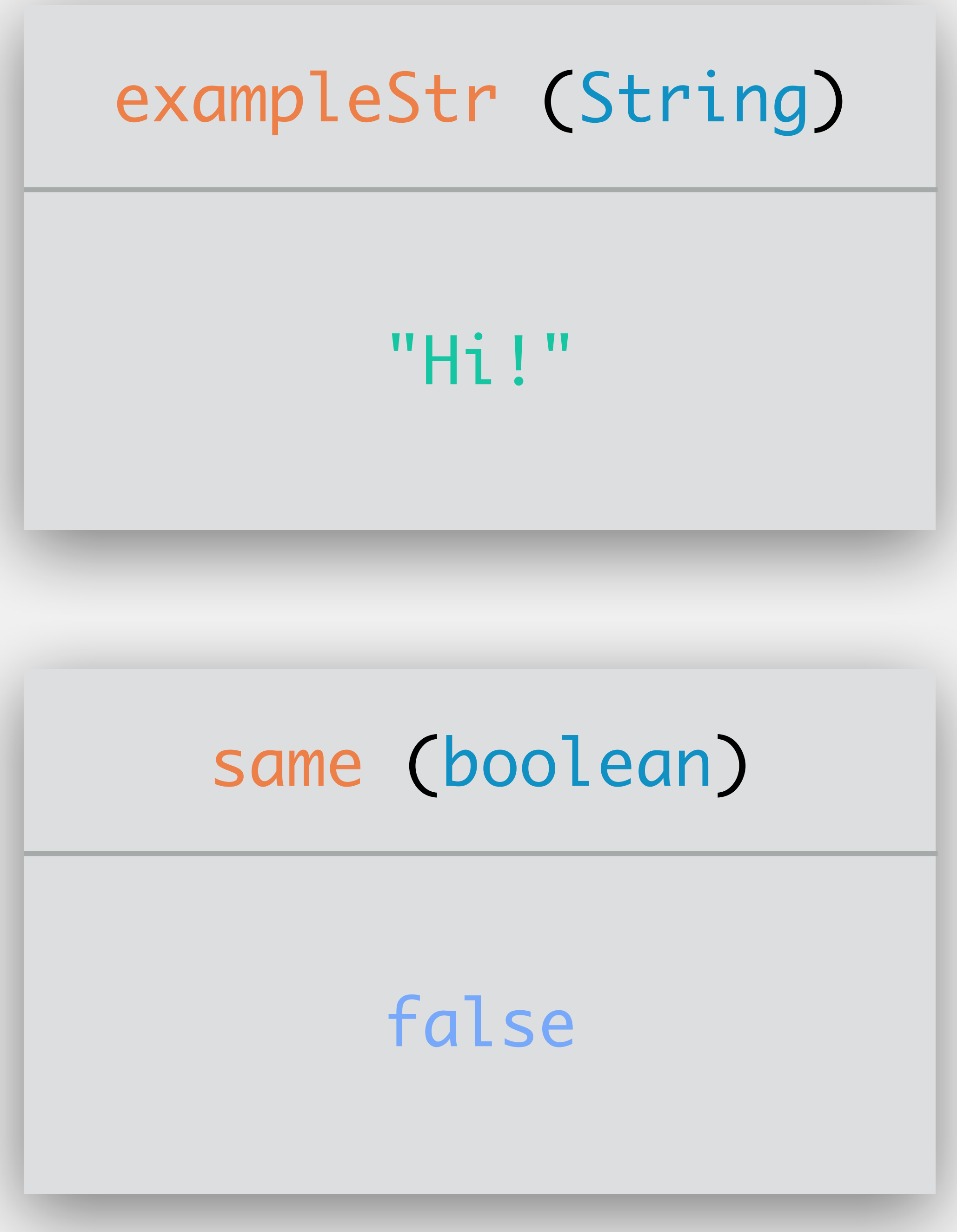
returns: a boolean value; true if the two Strings are the same, false if not

```
<String>.equals(<String>);
```

```
>String exampleStr = "Hi!";

>boolean same = exampleStr.equals("hi!");
>
```

memory



exampleStr (String)

"Hi!"

same (boolean)

false

==

arguments: two String values to compare

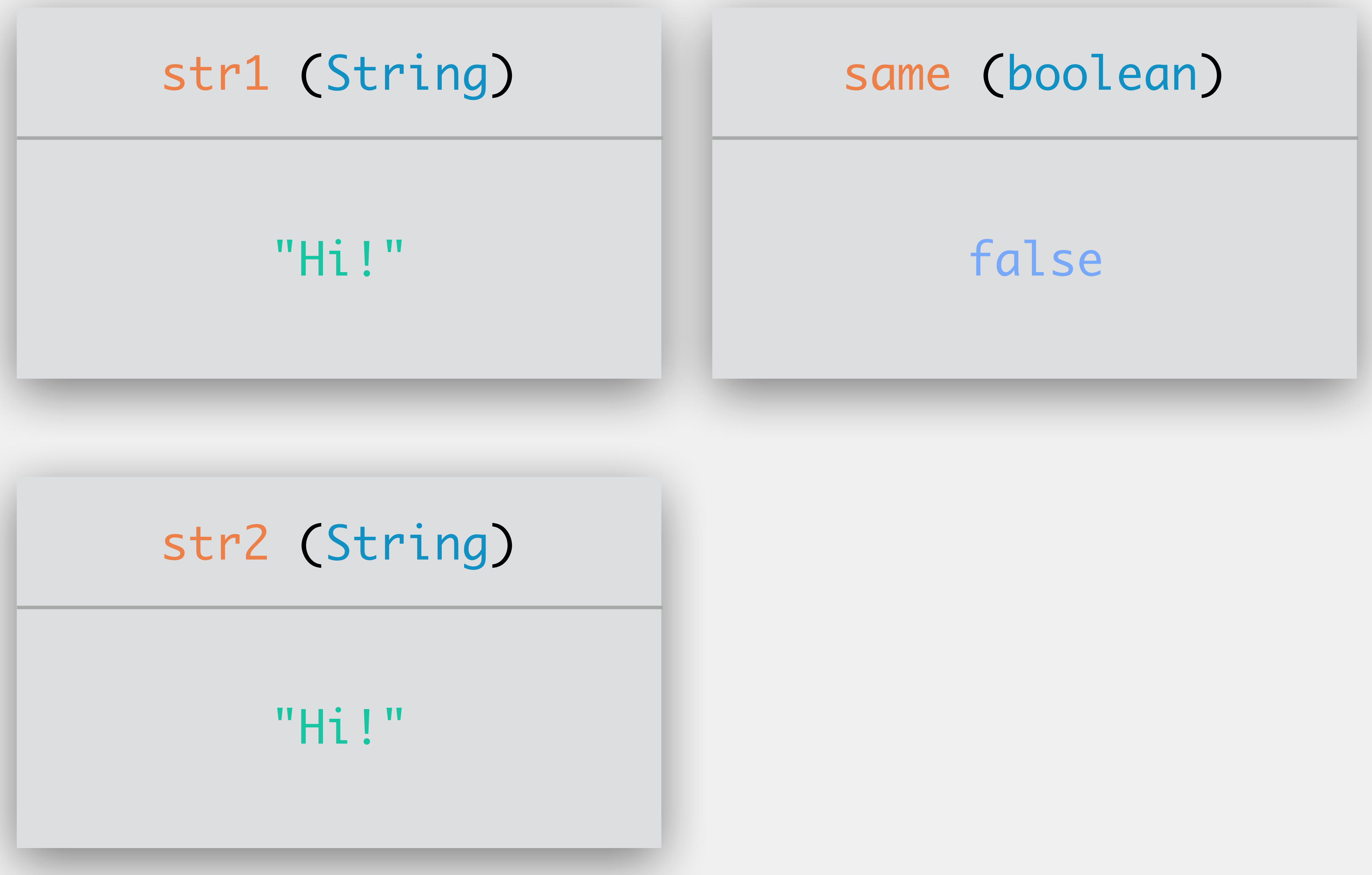
returns: a boolean value; true if the Strings are at the same memory location

```
<String> == <String>;
```

```
> String str1 = "Hi!", str2 = "Hi!";

> boolean same = str1 == str2;
>
```

memory



str1 (String)

"Hi!"

same (boolean)

false

str2 (String)

"Hi!"

== vs equals()

Primitive data types (boolean, char, int, double, …)

- always use ==
- will check to see if the two are the same value
- .equals() does not exist for primitive data types

Class data types (String, …)

- will almost always use .equals()
    - will check to see if the content of the two objects is the same
    - we can define what equality means!
- == will check if the memory location of the two objects is the same

Searching & Sorting

Data structures can contain multiple pieces of information in a single place

Often want to manipulate these

- searching
- sorting

Searching An Array

Examine each index until we find what we are looking for

Searching Modifications

- Know there are one vs many occurrences
    - one: can stop after it’s found
    - many: must continue until the end of the loop
- Searching for first vs all occurrences
    - one: can stop after the first is found
    - many: must continue until the end of the loop

Sorting An Array

Numerous sorting algorithms available

- many algorithms + their efficiency (i.e., *complexity*) will be discussed in 340

In this class

- selection sort
- insertion sort

Selection Sort

Considered one of the classic sorting algorithms

Very simple, but very inefficient

- will do the job for this class

Basic premise:

- scans through the array multiple times, looking for the next smallest element each time
- moves the smallest element to the front of the array

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

- initially, everything is unsorted

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

Repeat

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

8 3 2 5 9 7

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

8 3 2 5 9 7

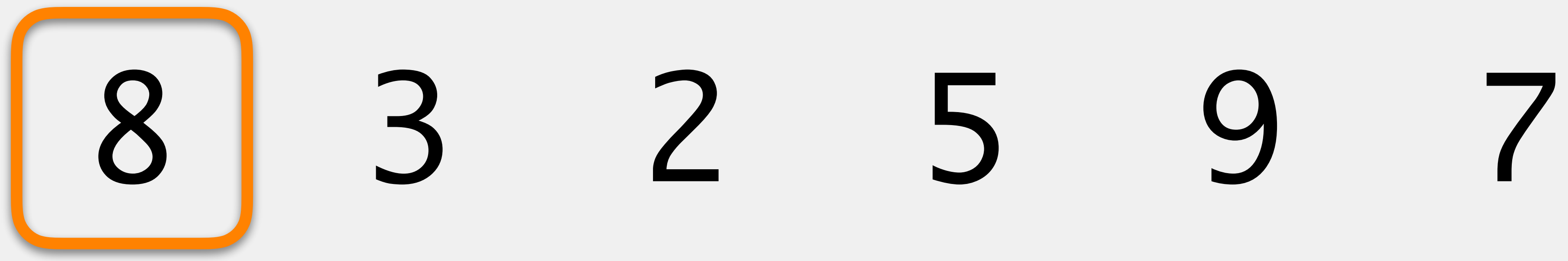
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



8 3 2 5 9 7

```
smallestIndex = 0
```

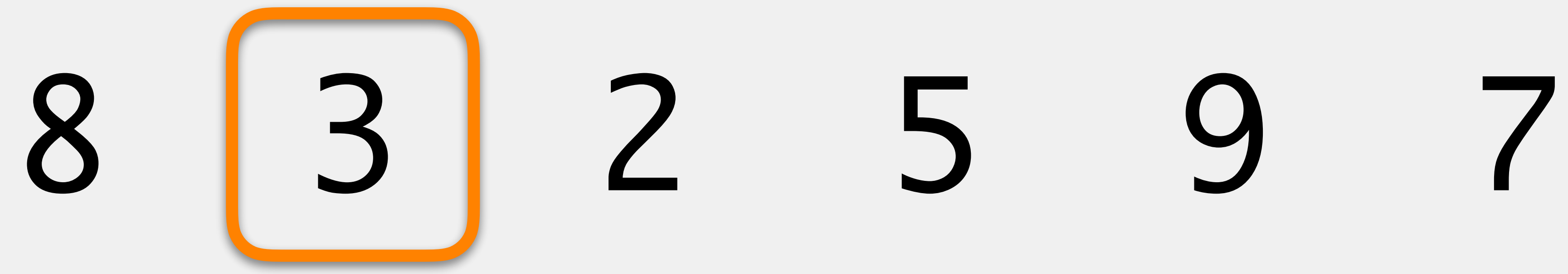
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



8 3 2 5 9 7

```
smallestIndex = 1
```

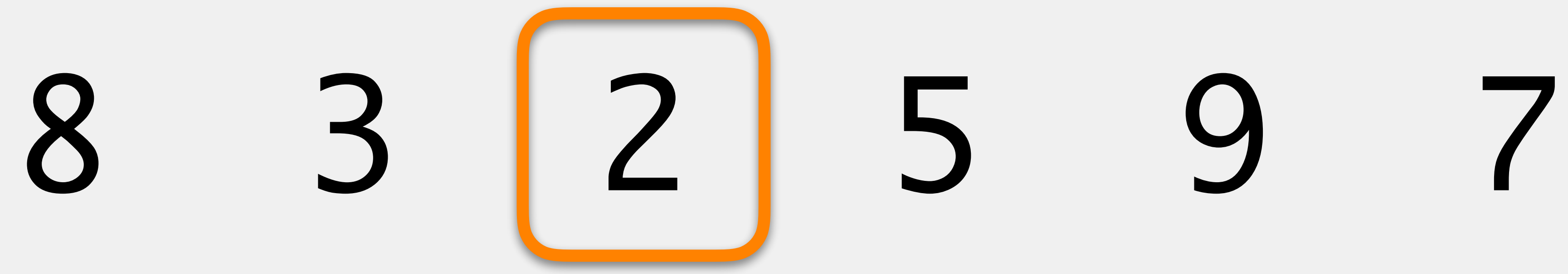
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



8 3 2 5 9 7

```
smallestIndex = 2
```

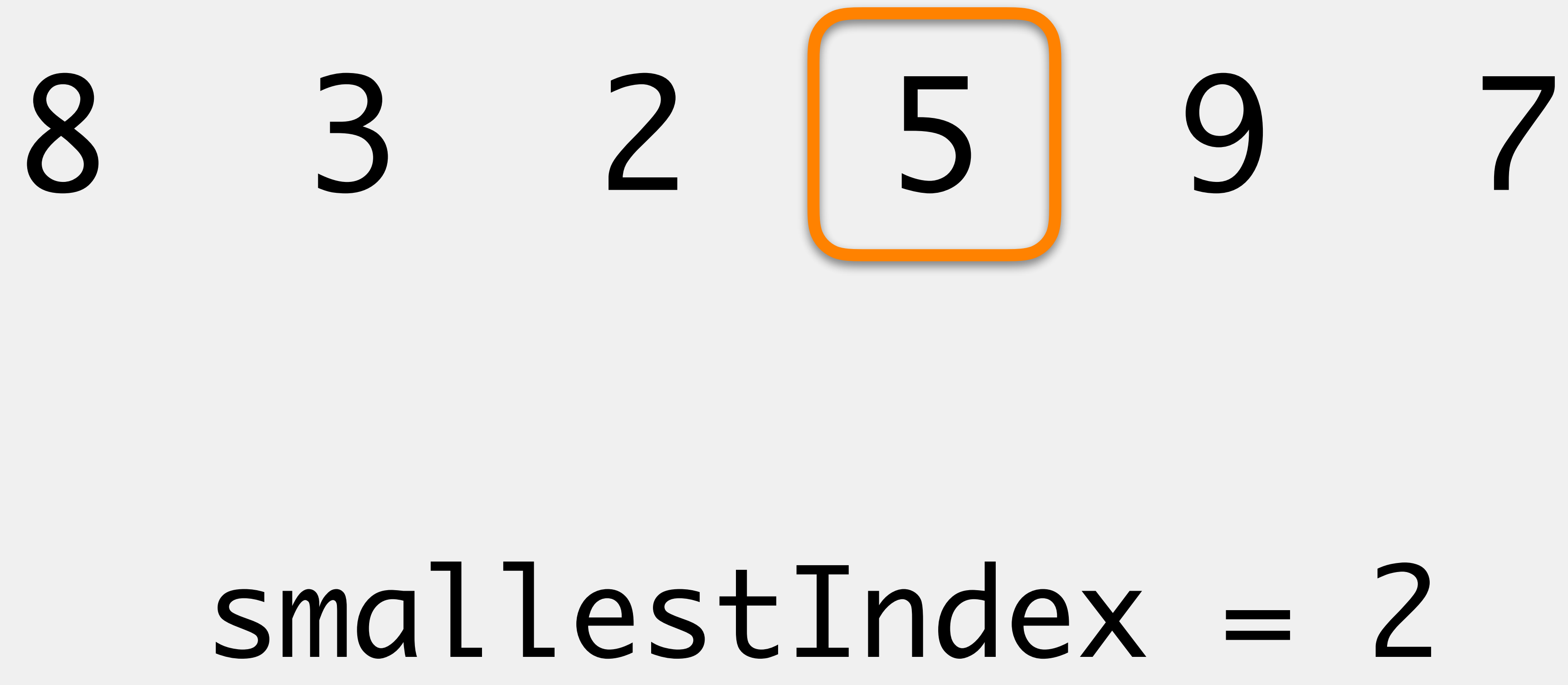
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



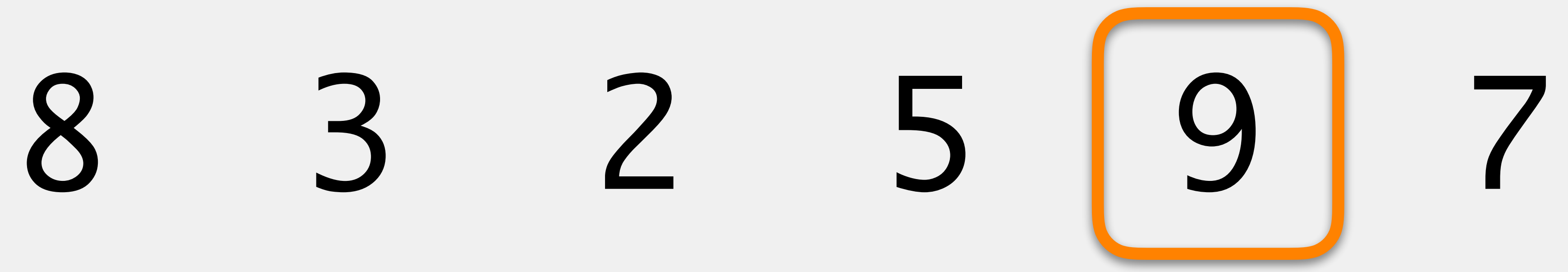
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



8 3 2 5 9 7

```
smallestIndex = 2
```

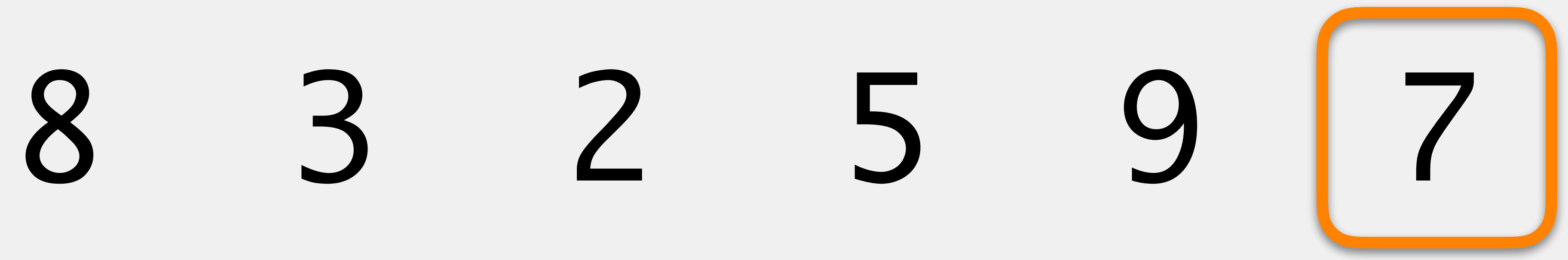
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



8 3 2 5 9 7

```
smallestIndex = 2
```

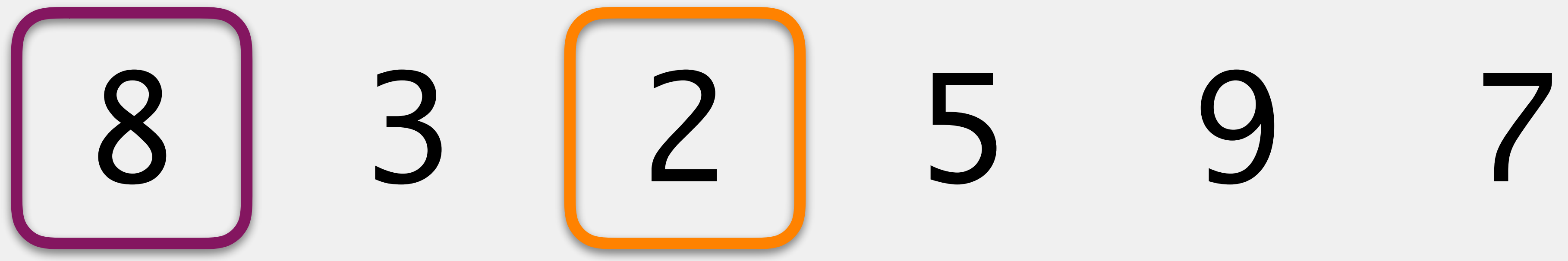
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



8 3 2 5 9 7

```
smallestIndex = 2
```

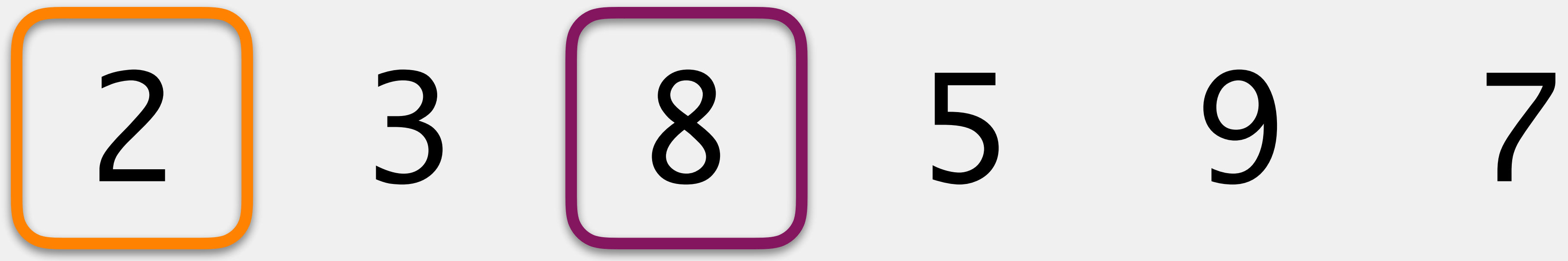
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

```
smallestIndex = 2
```

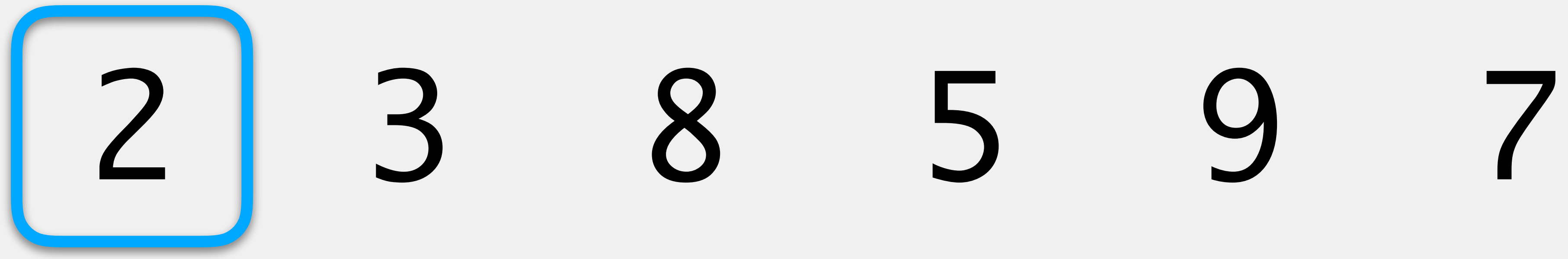
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

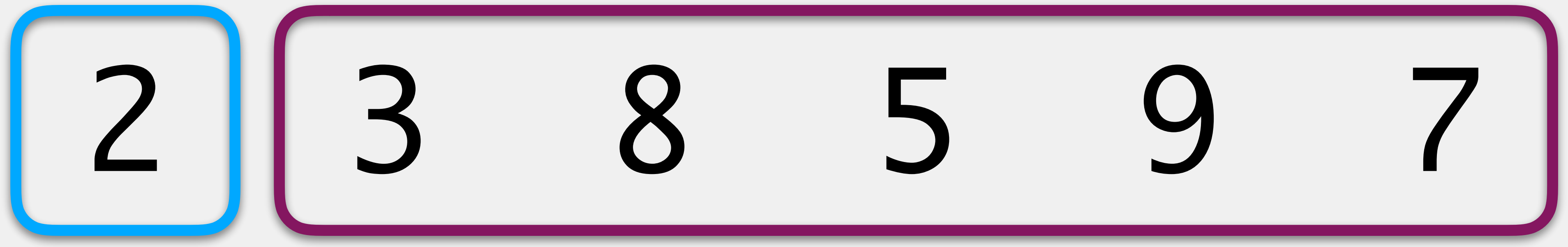
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2

3 8 5 9 7

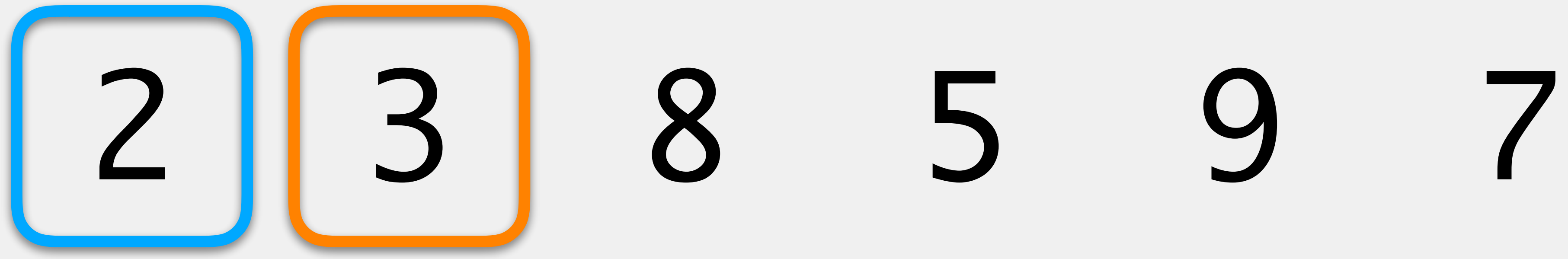
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

`smallestIndex = 1`

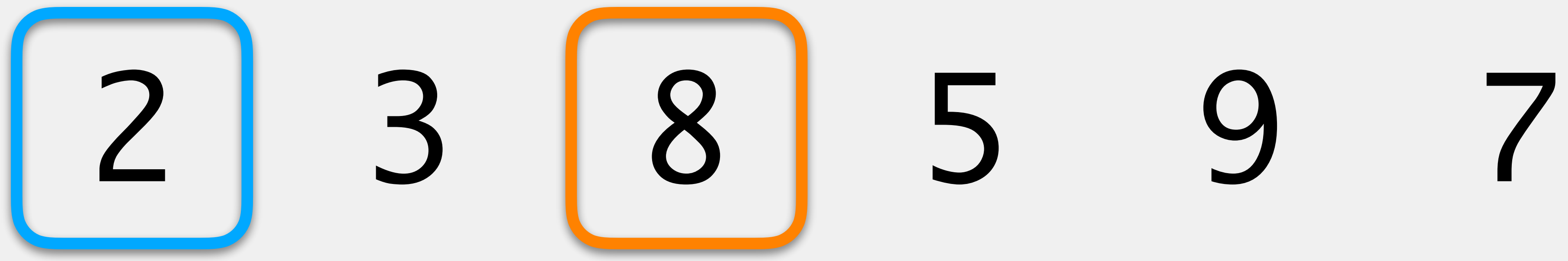
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



```
smallestIndex = 1
```

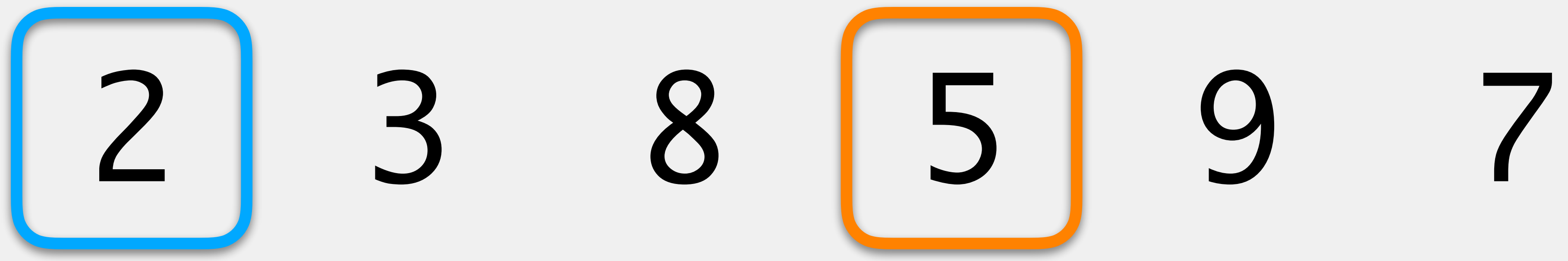
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

```
smallestIndex = 1
```

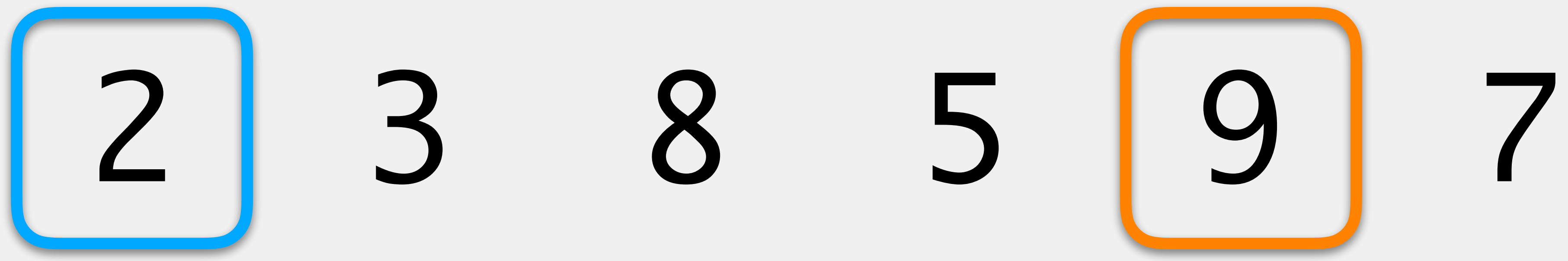
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

```
smallestIndex = 1
```

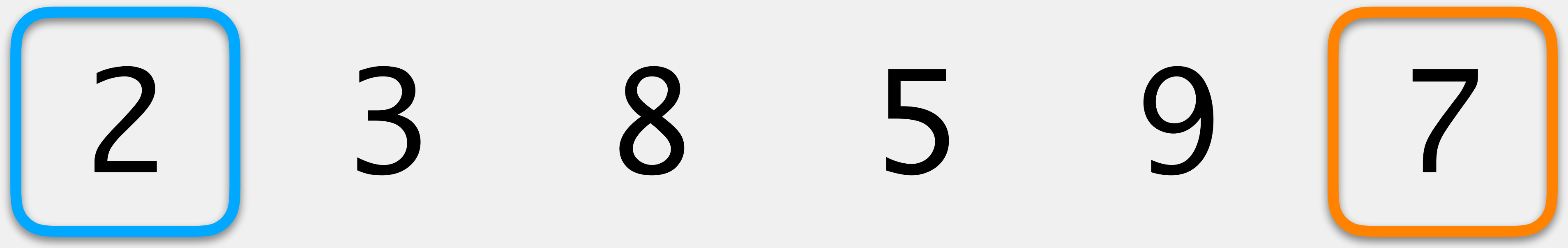
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

```
smallestIndex = 1
```

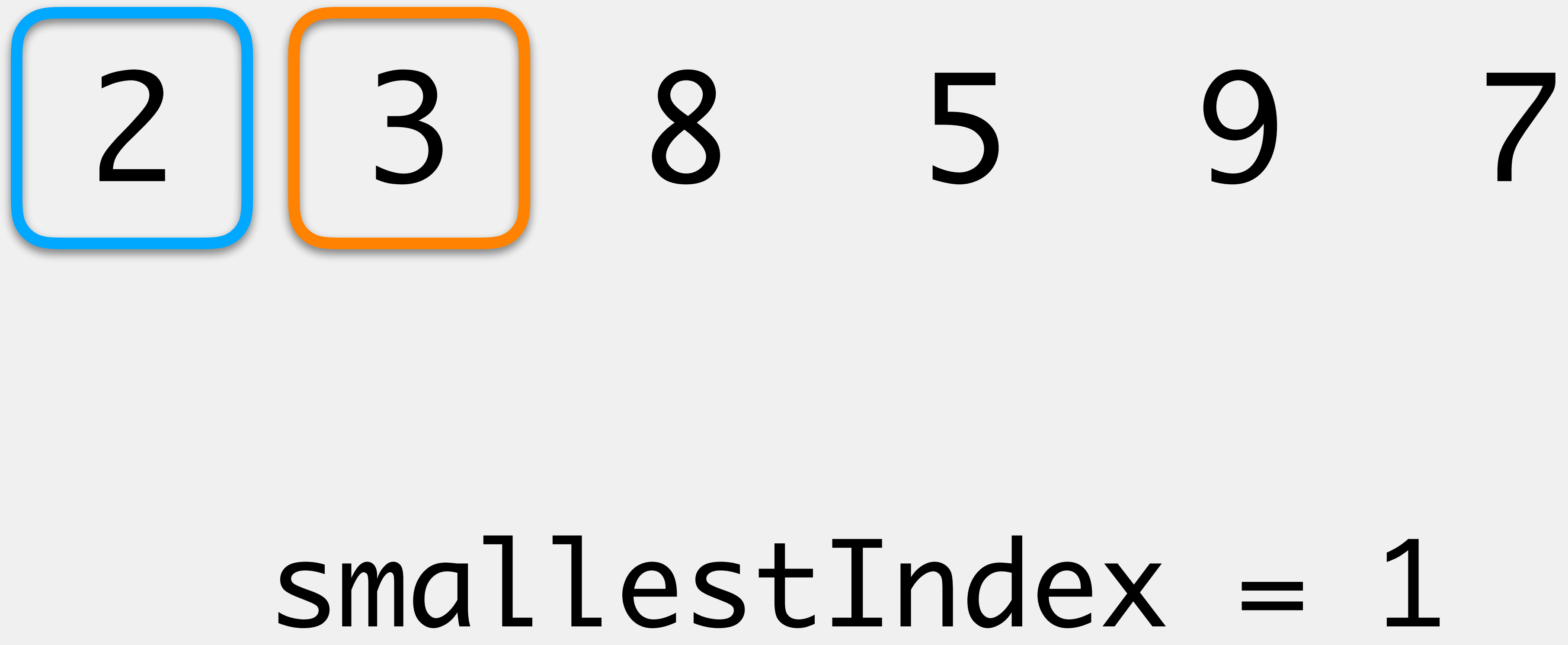
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

`smallestIndex = 1`

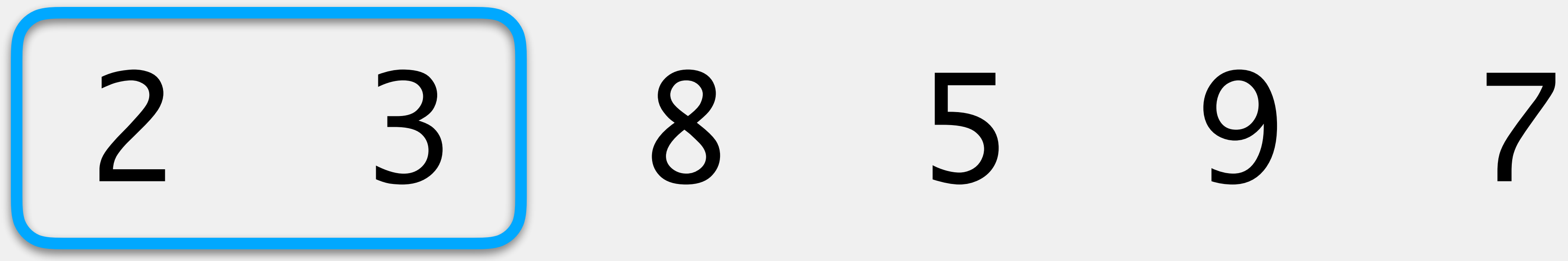
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

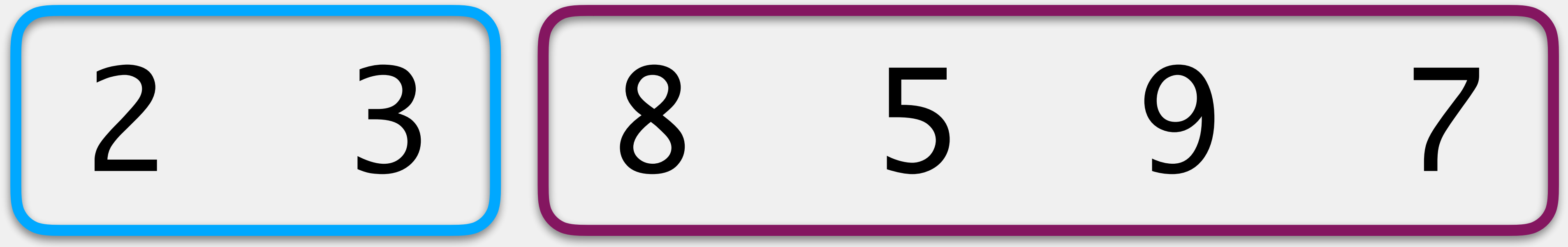
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3

8 5 9 7

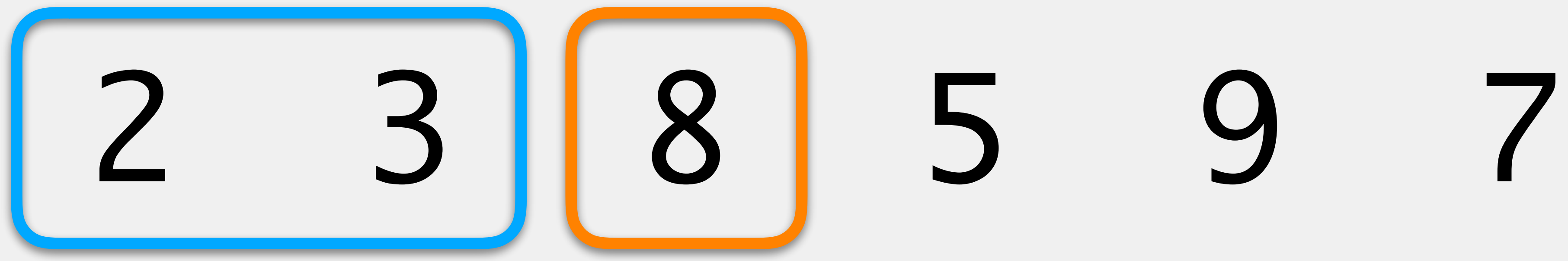
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

smallestIndex = 2

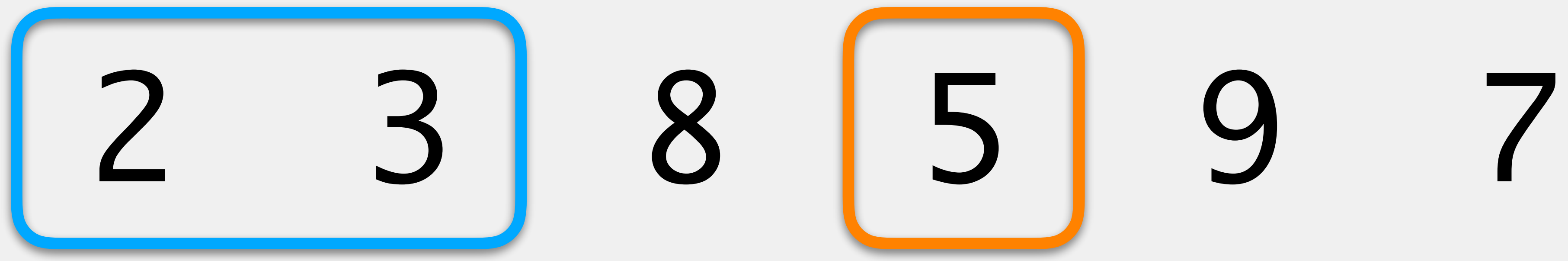
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

```
smallestIndex = 3
```

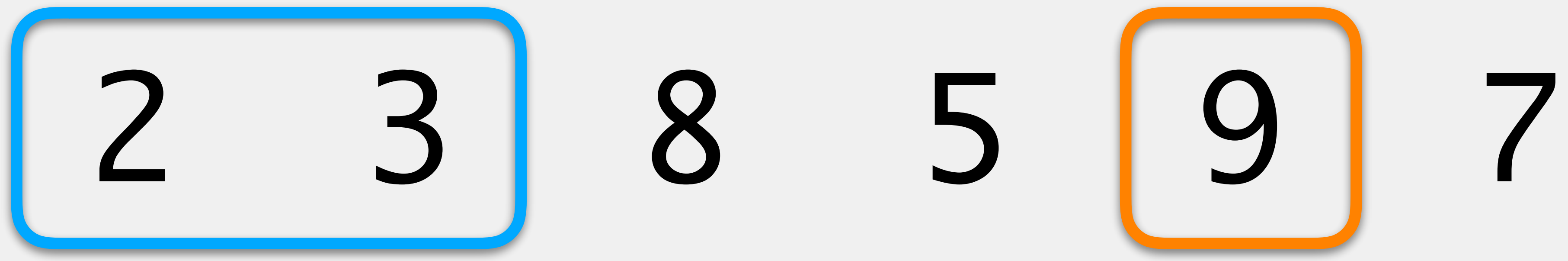
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

```
smallestIndex = 3
```

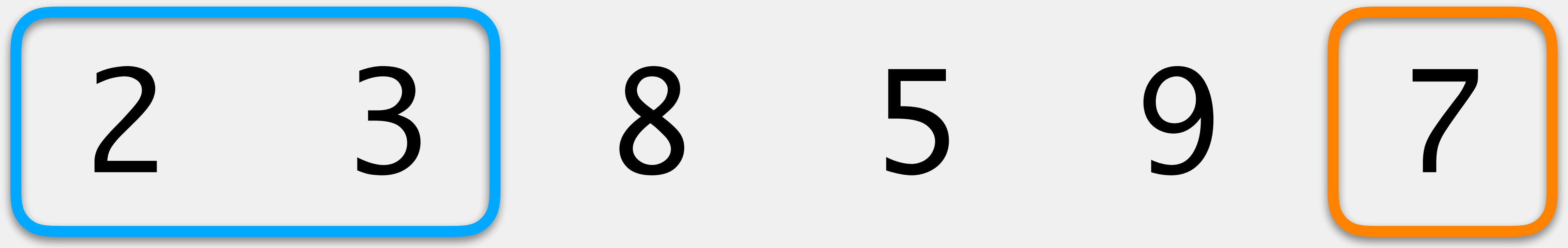
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



```
smallestIndex = 3
```

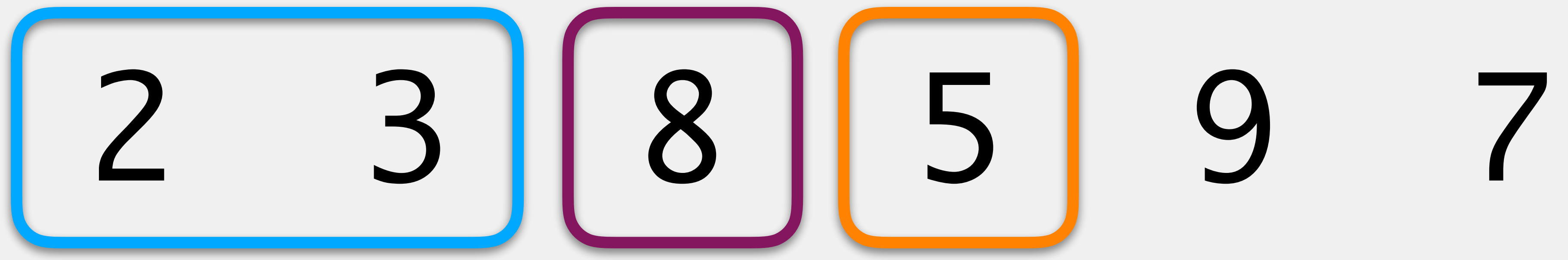
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

`smallestIndex = 3`

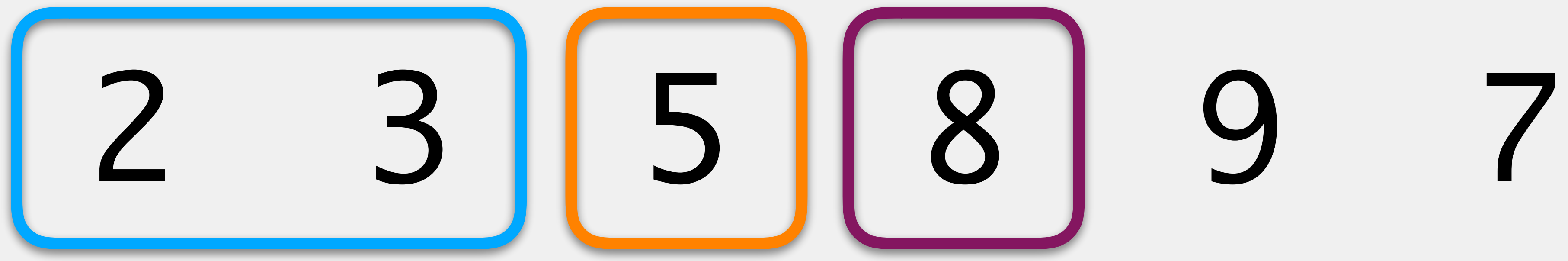
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

`smallestIndex = 3`

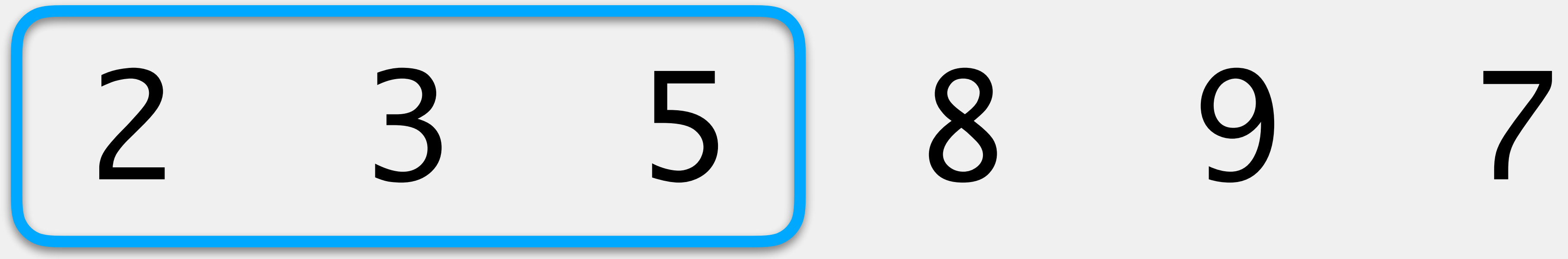
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

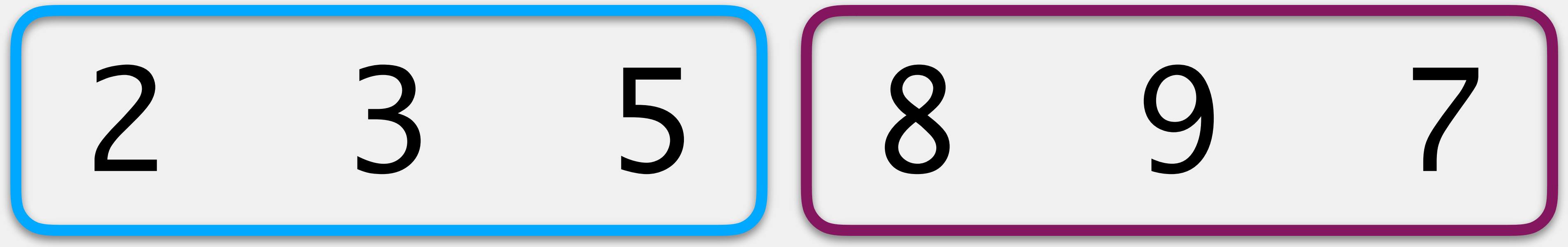
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



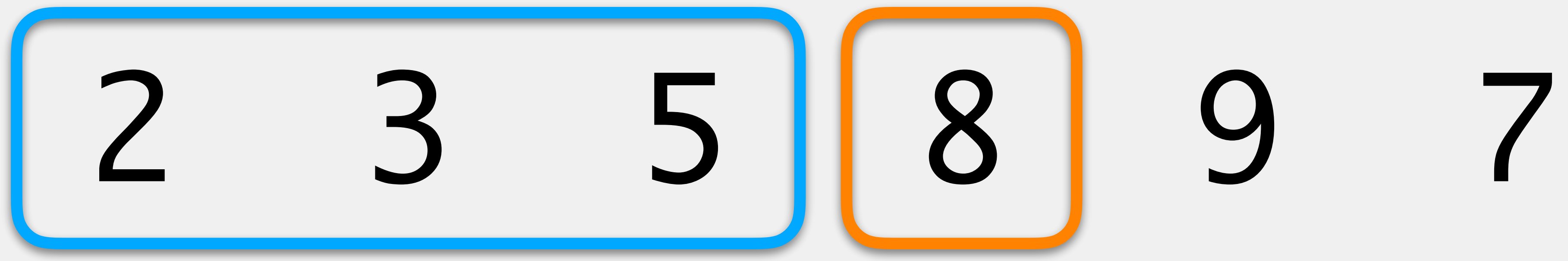
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



```
smallestIndex = 3
```

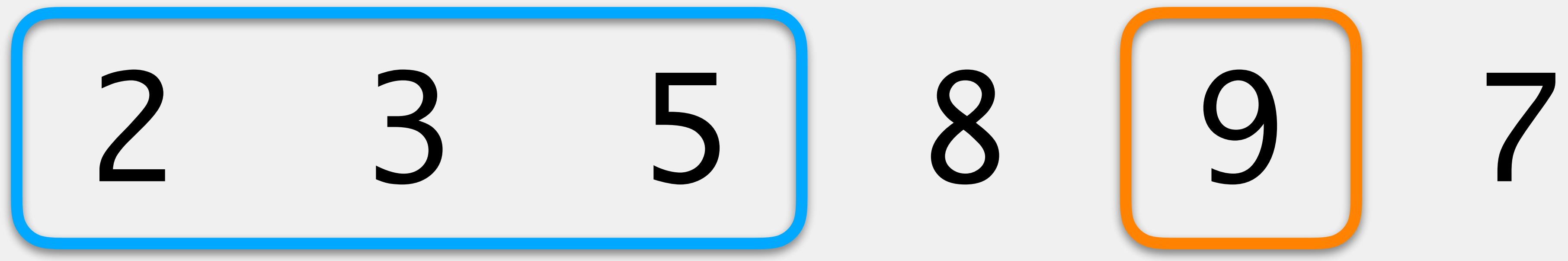
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

`smallestIndex = 3`

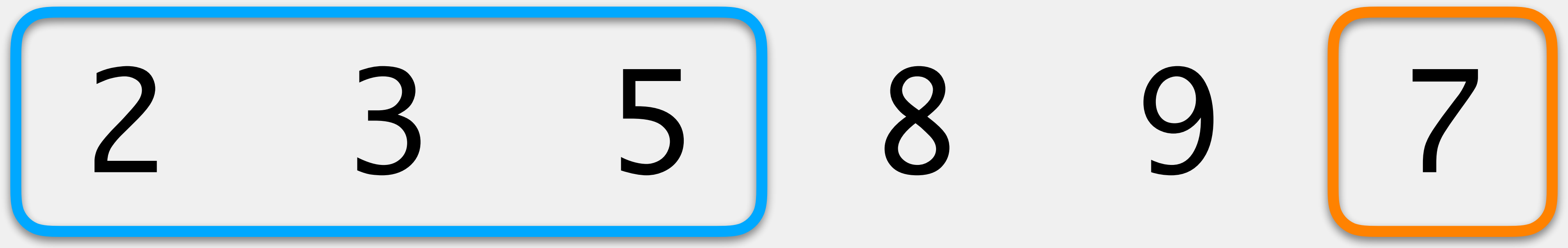
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

`smallestIndex = 5`

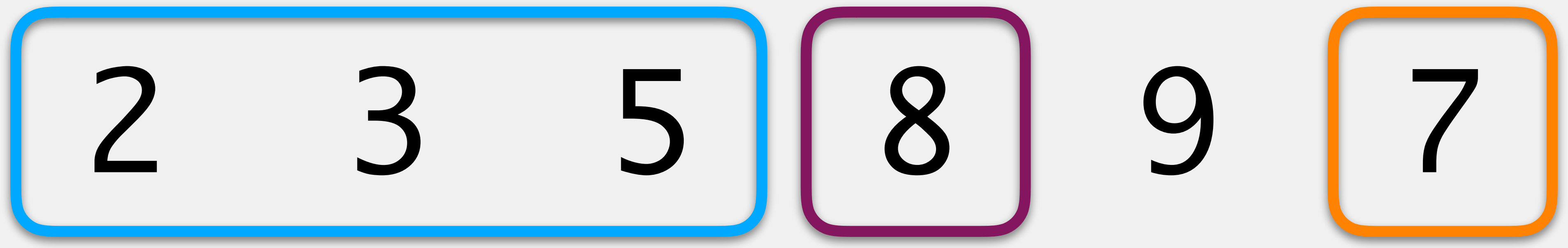
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

```
smallestIndex = 5
```

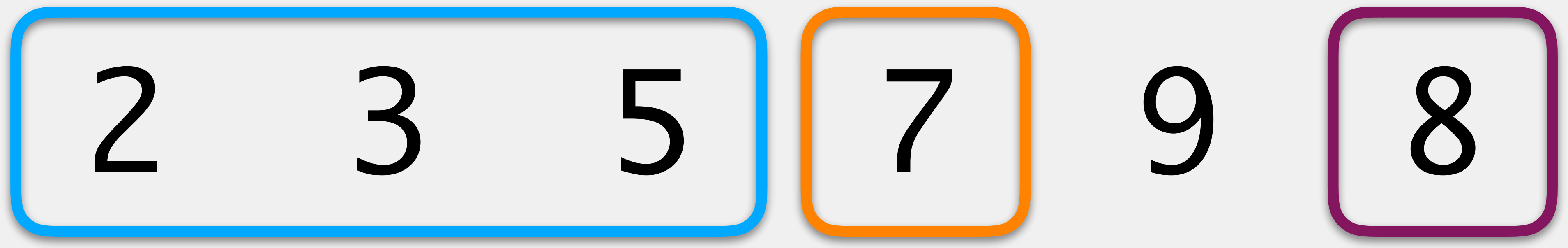
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



```
smallestIndex = 5
```

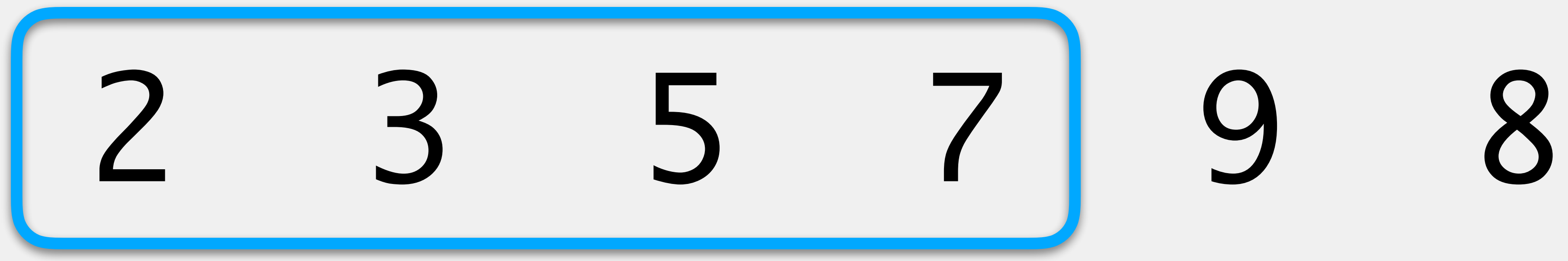
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 7 9 8

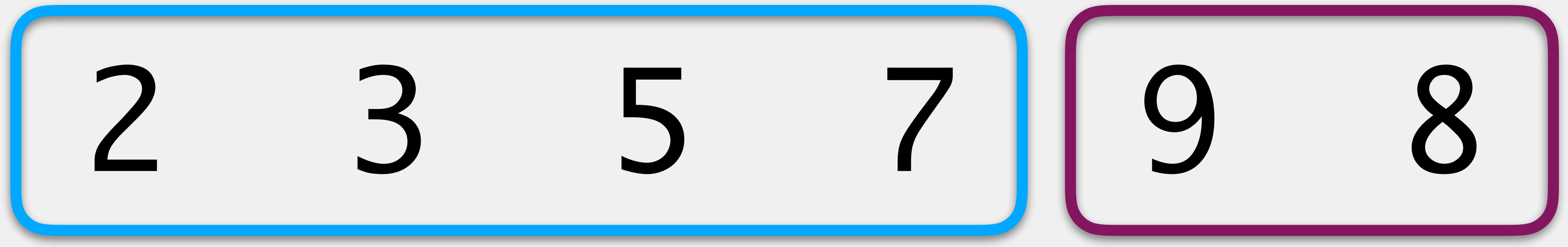
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 7

9 8

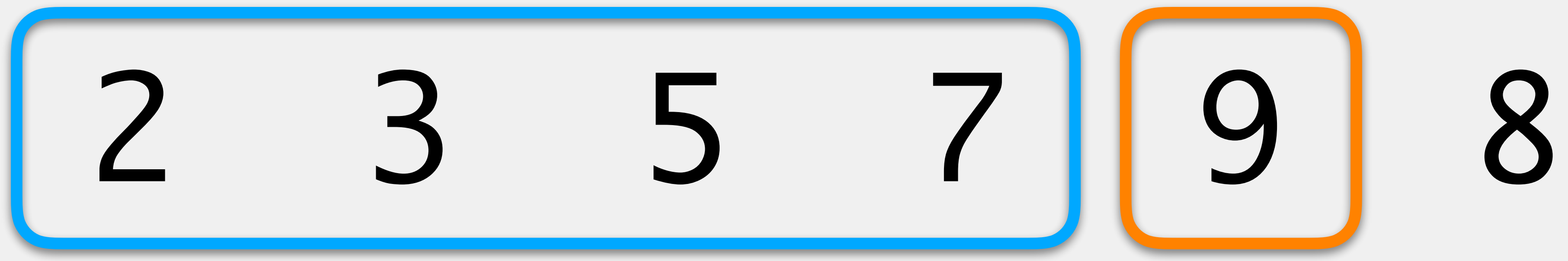
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



smallestIndex = 4

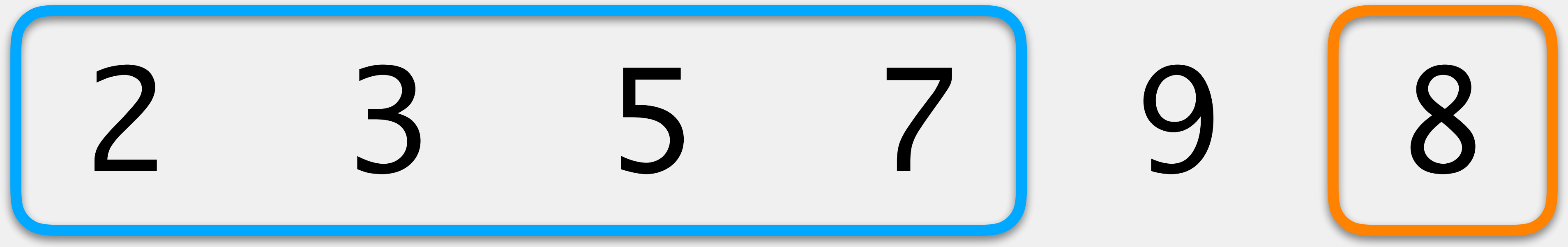
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 7 9 8

```
smallestIndex = 5
```

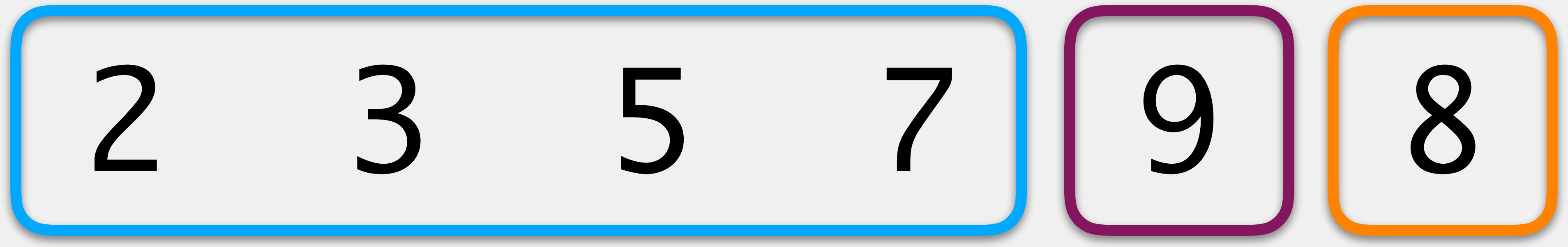
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 7 9 8

```
smallestIndex = 5
```

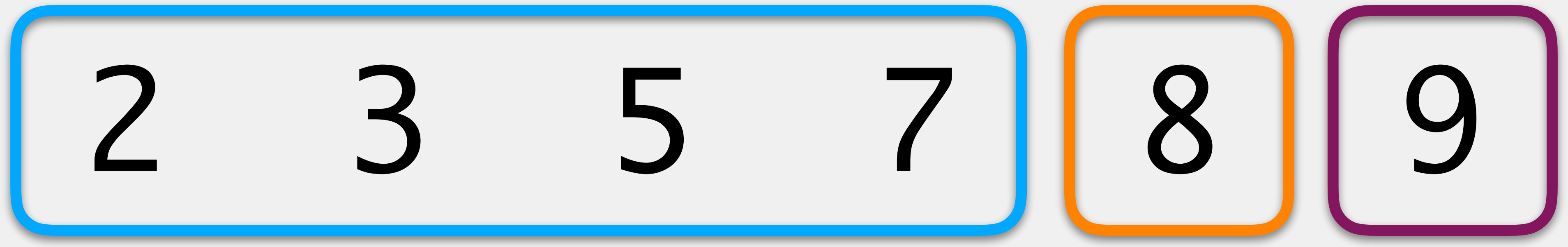
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



```
smallestIndex = 5
```

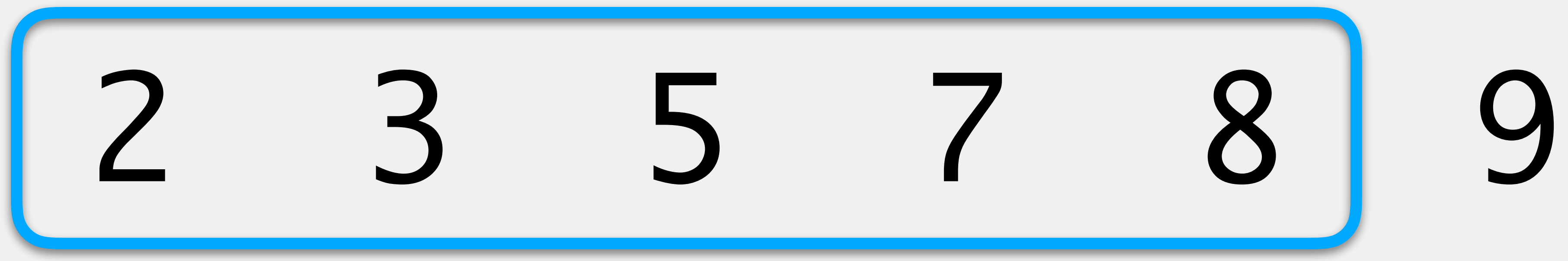
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 7 8 9

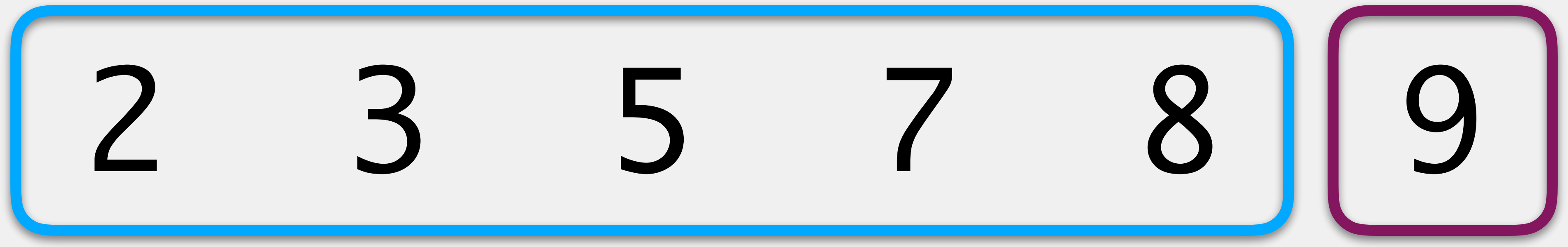
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



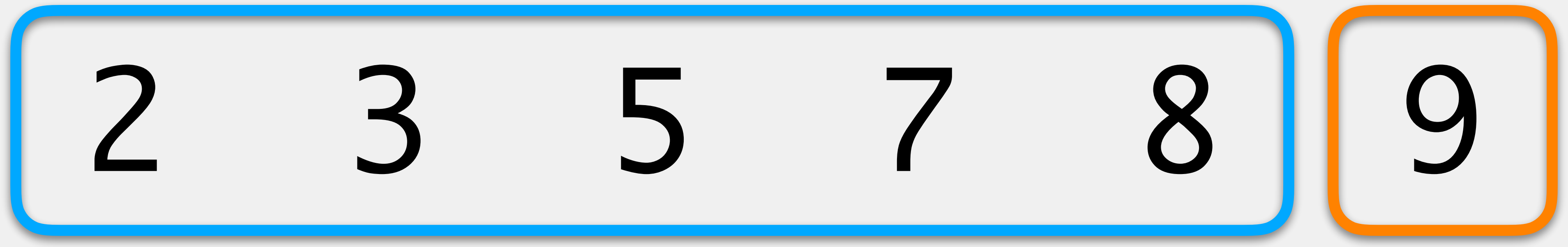
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



```
smallestIndex = 5
```

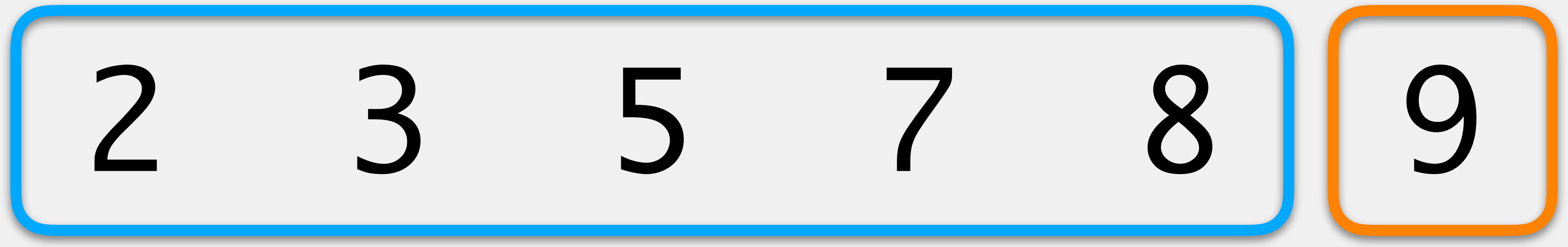
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



```
smallestIndex = 5
```

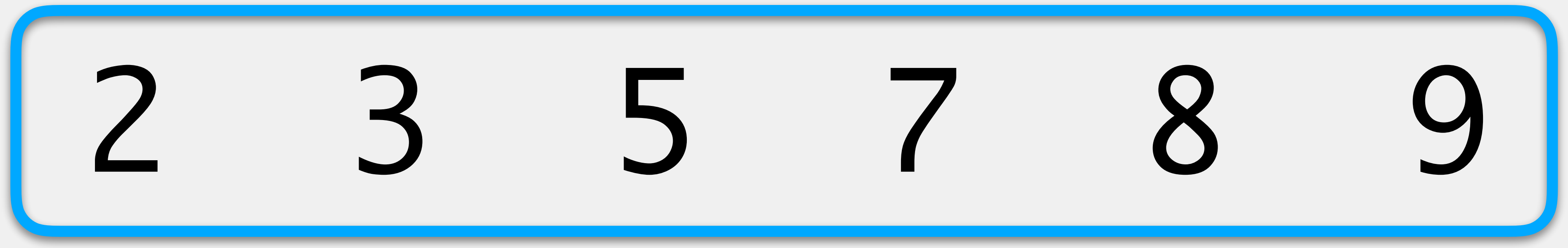
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 7 8 9

Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 5 7 8 9

Insertion Sort

Considered one of the classic sorting algorithms

Very simple, but very inefficient

- will do the job for this class

Basic premise:

- scans through the array multiple times, looking at the next unsorted element
- moves that unsorted element into a sorted place in the final list

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

- initially, first element is sorted, everything else is unsorted

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

Repeat

Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

8 3 2 5 9 7

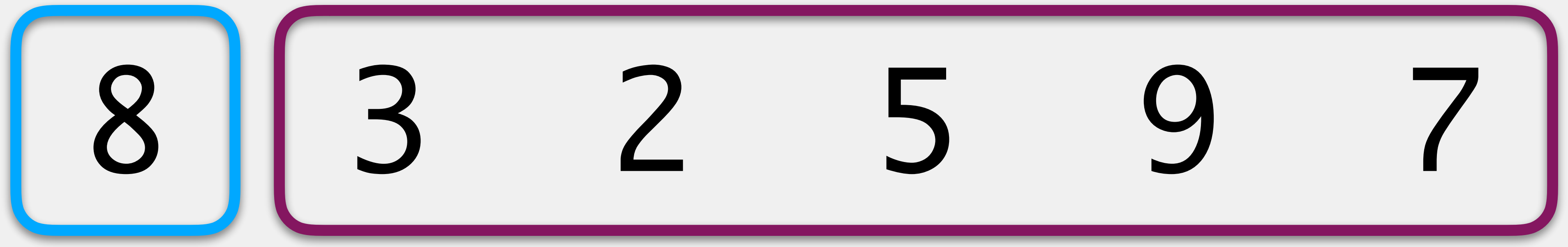
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



8 3 2 5 9 7

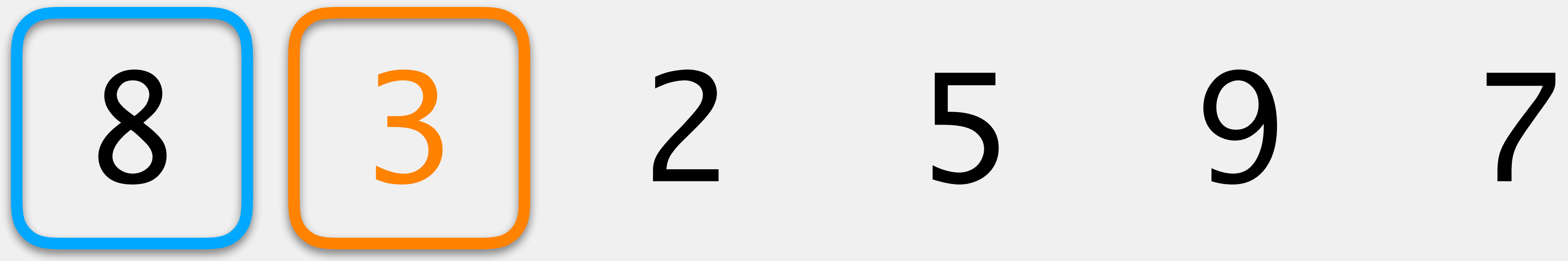
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



8 3 2 5 9 7

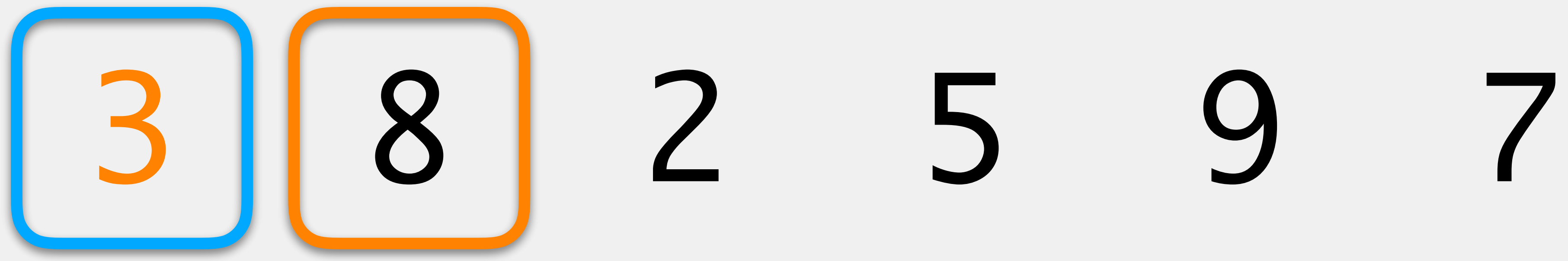
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



3 8 2 5 9 7

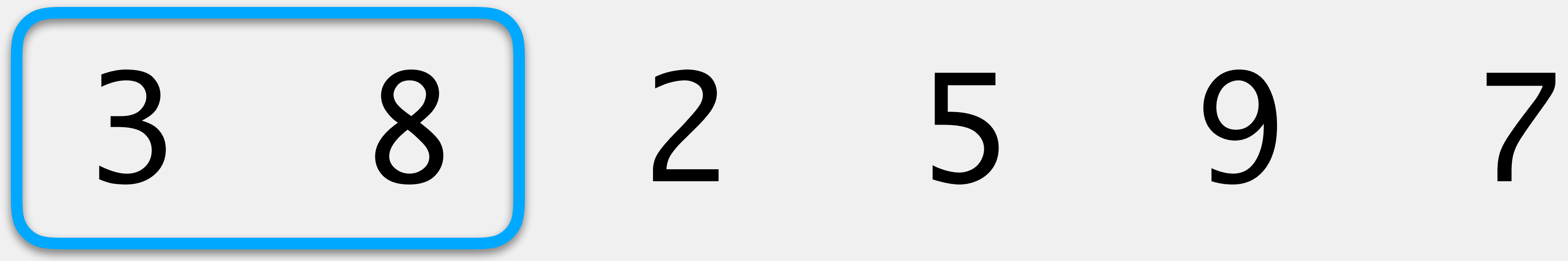
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



3 8 2 5 9 7

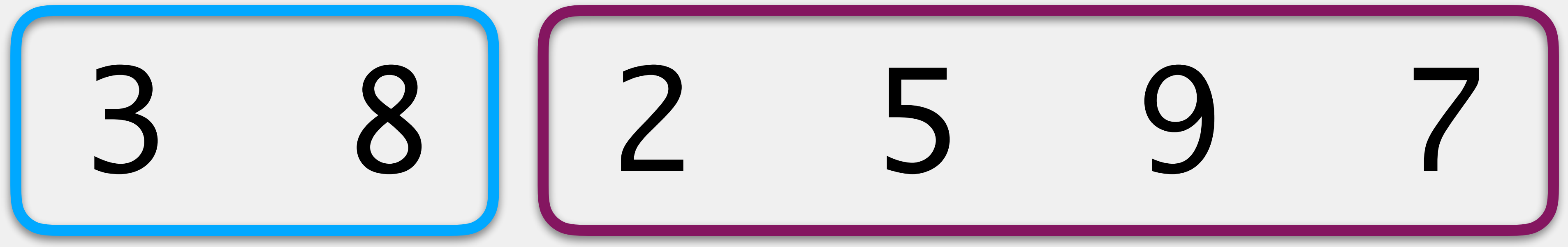
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



3 8

2 5 9 7

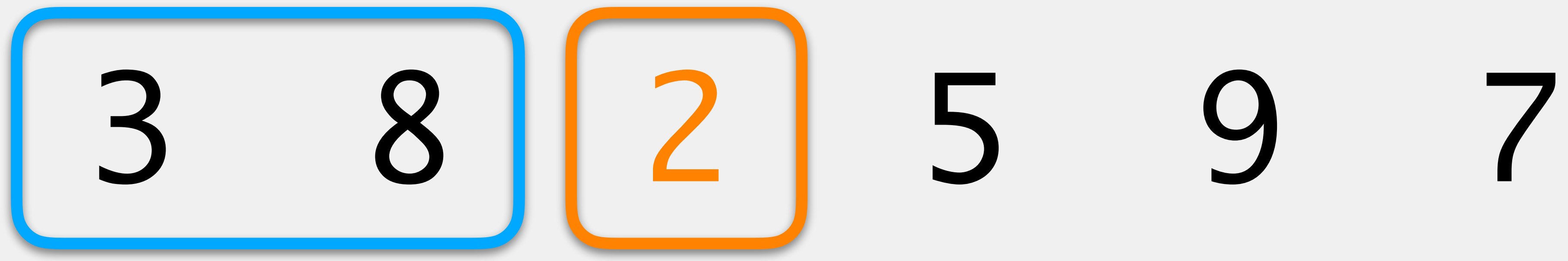
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



3 8 2 5 9 7

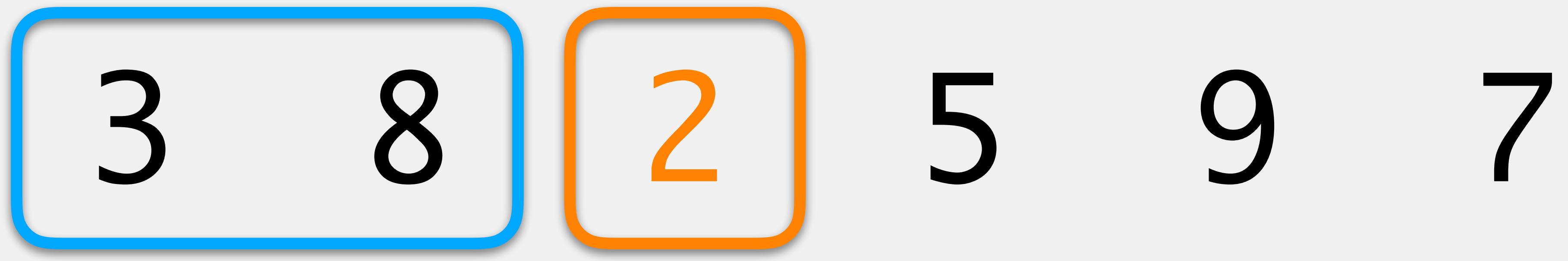
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



3 8 2 5 9 7

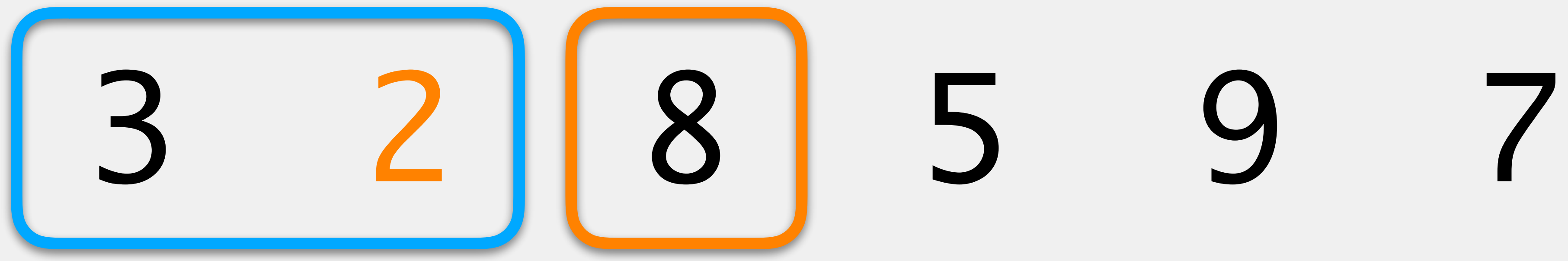
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



3 2 8 5 9 7

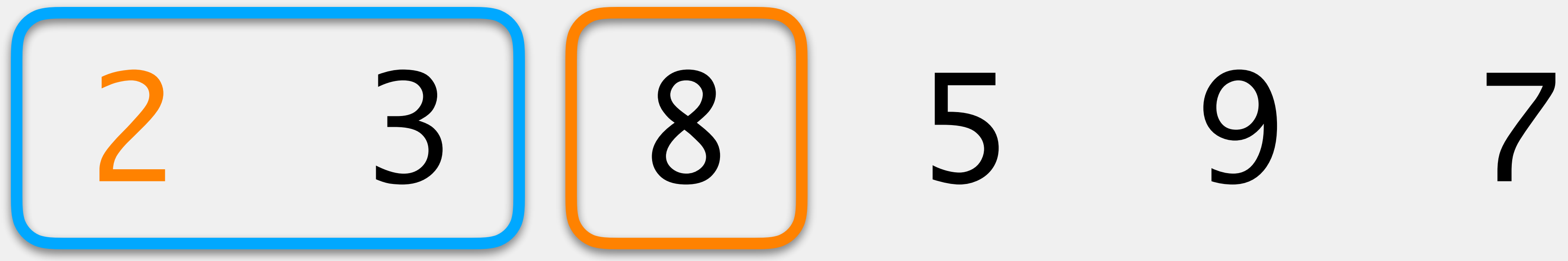
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

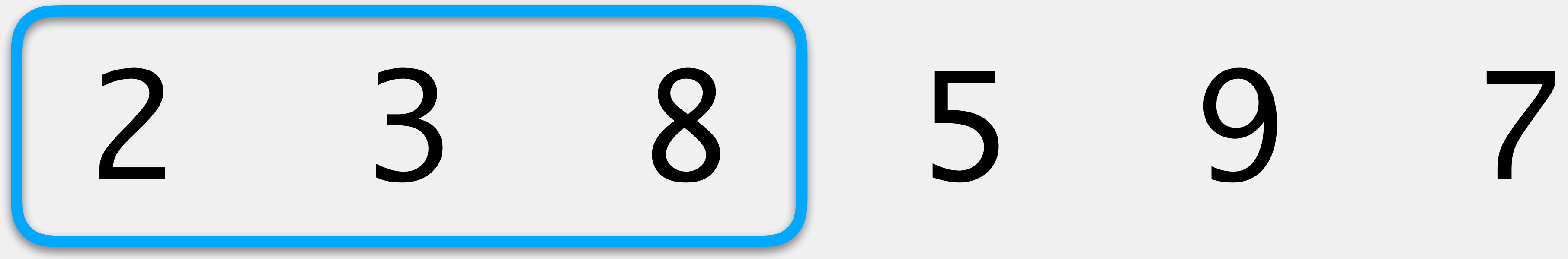
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

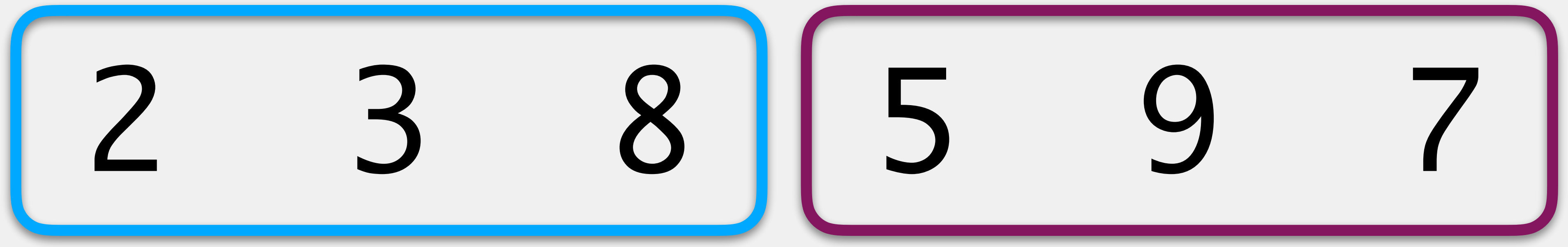
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8

5 9 7

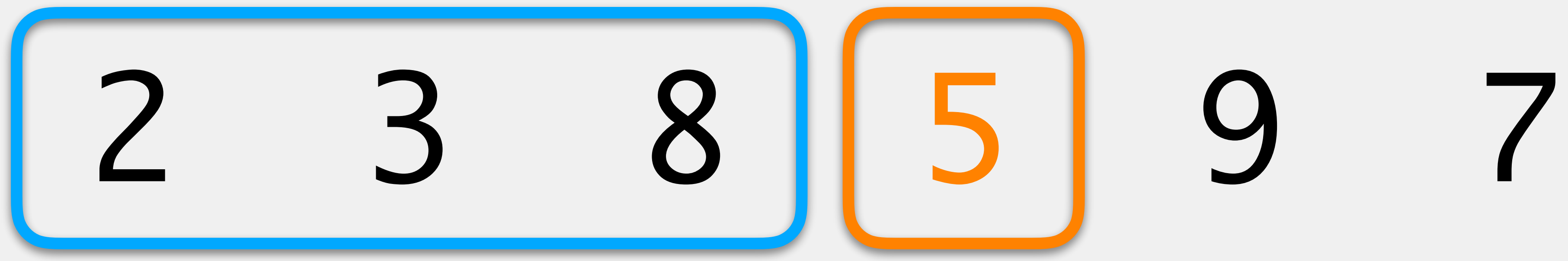
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

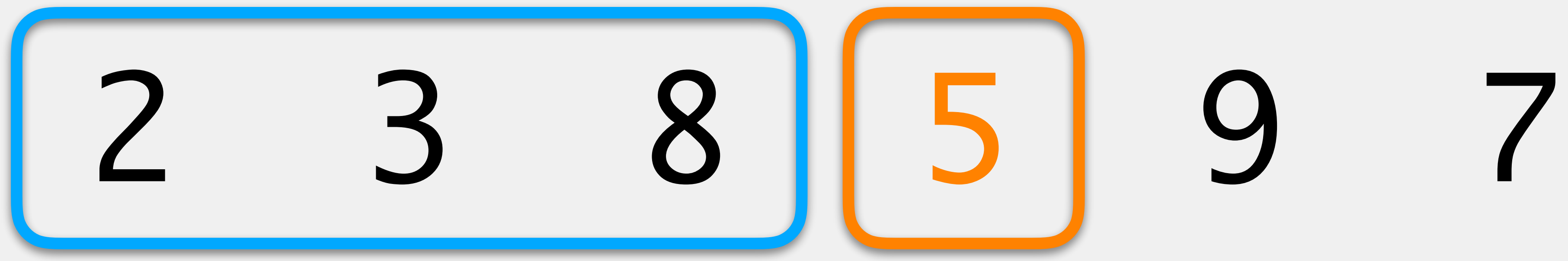
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 8 5 9 7

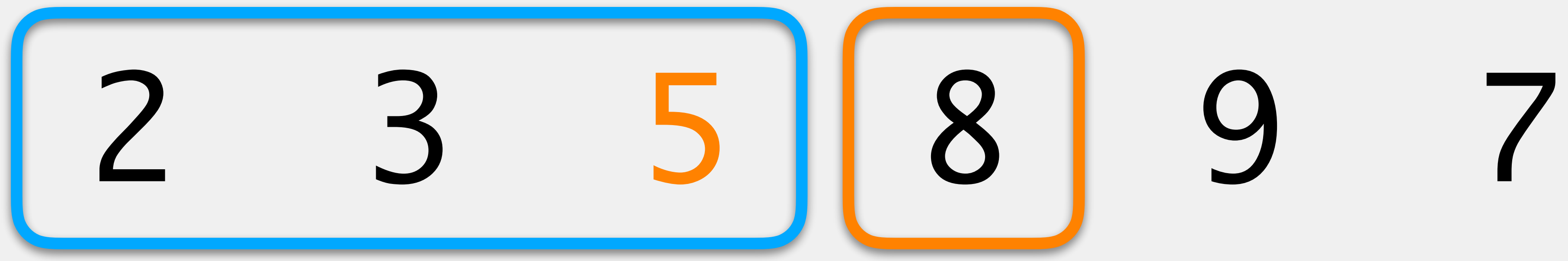
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

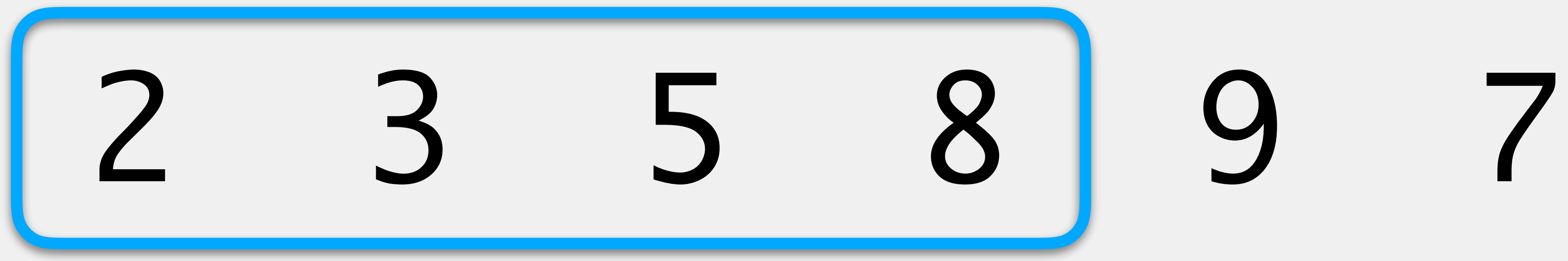
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

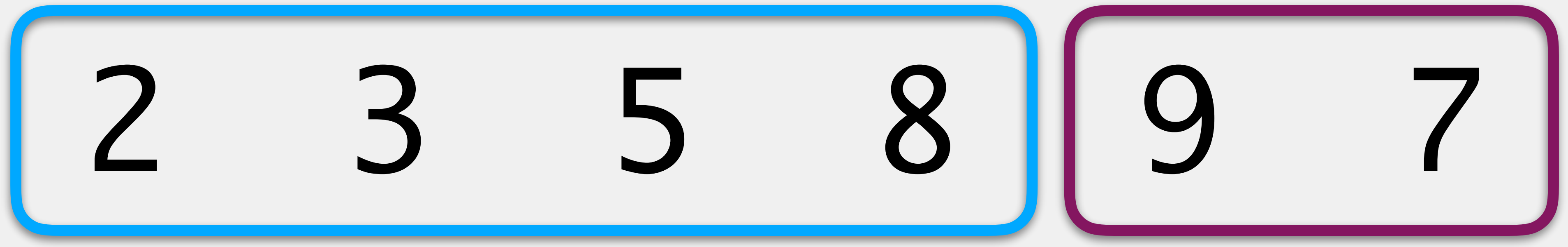
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8

9 7

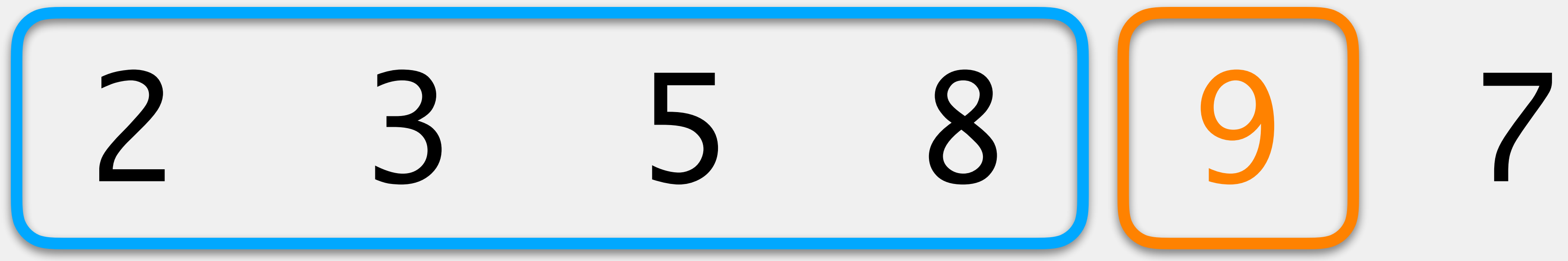
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

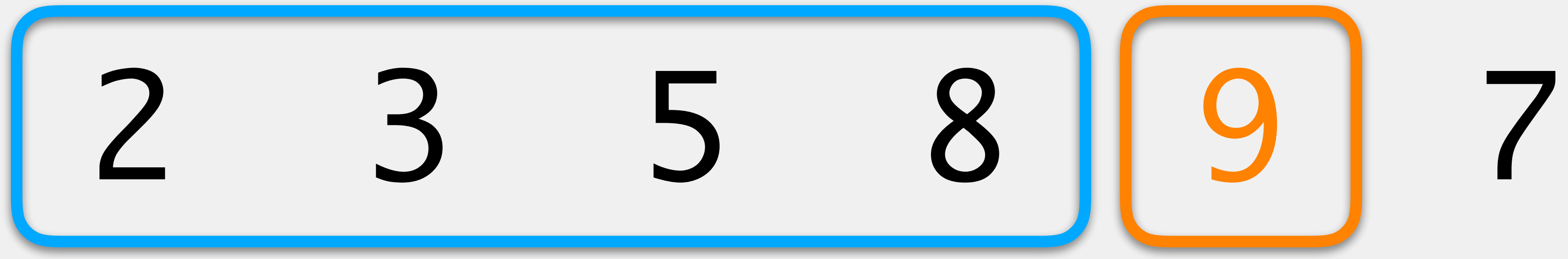
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

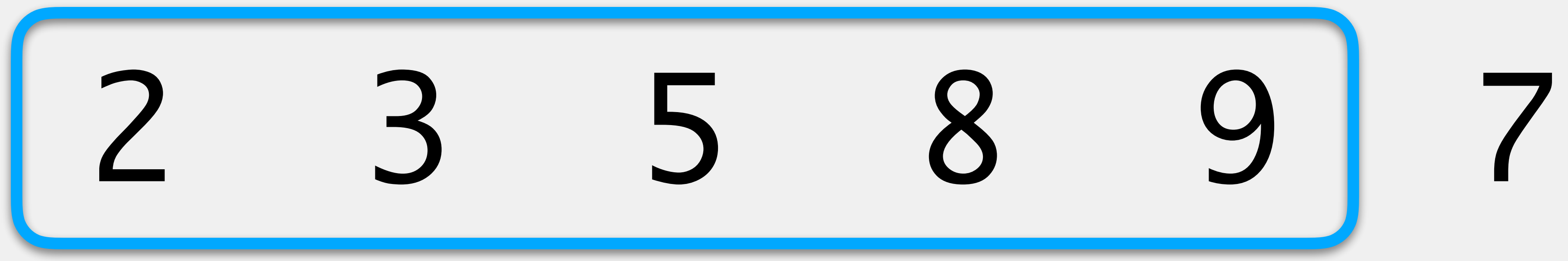
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

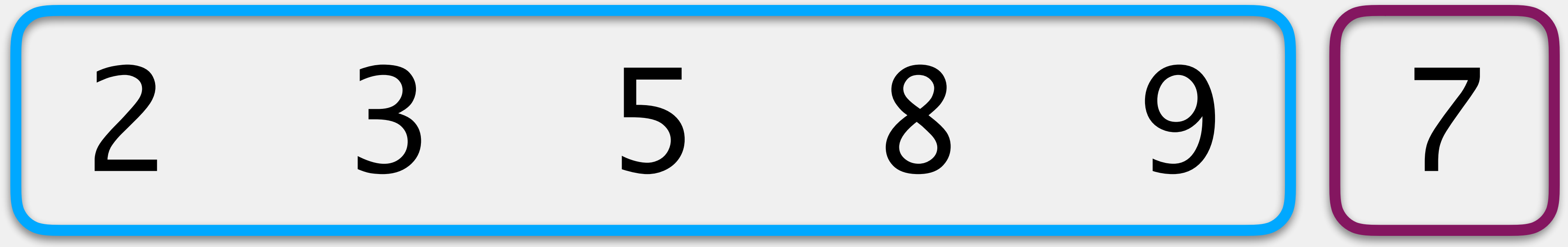
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9

7

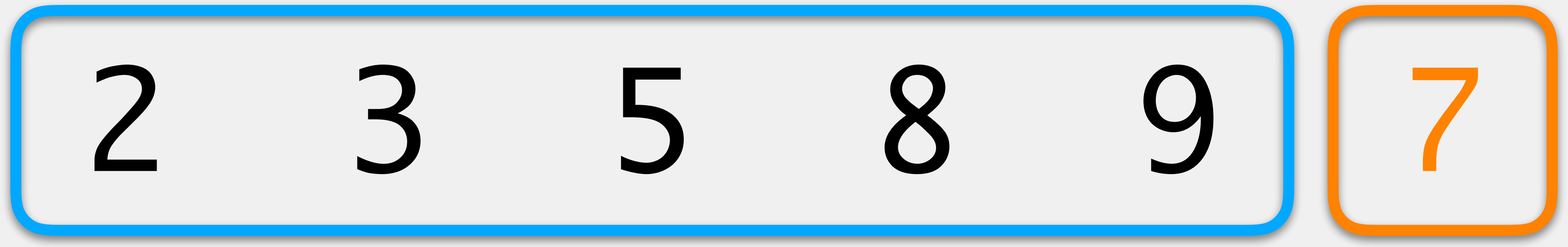
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

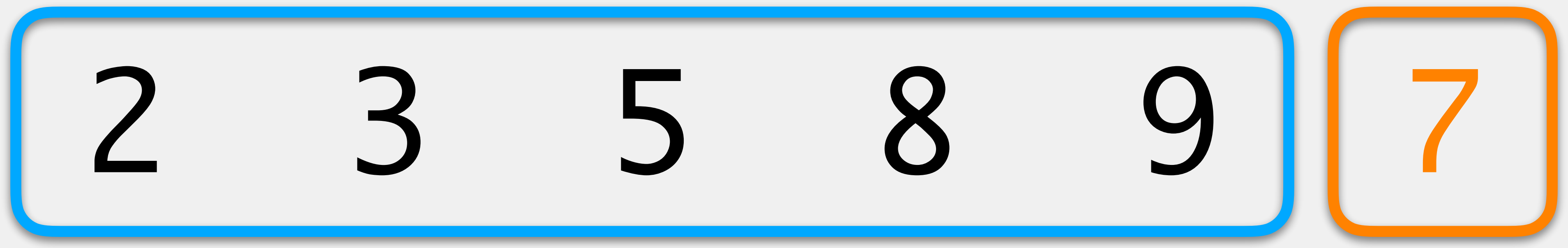
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 9 7

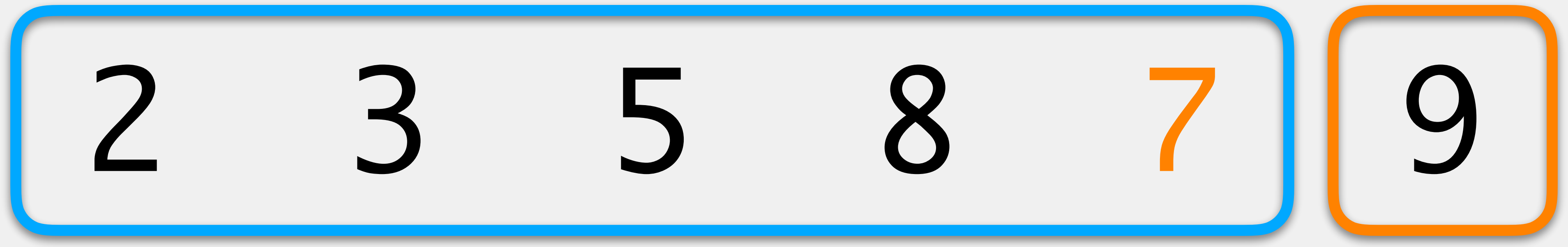
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 8 7 9

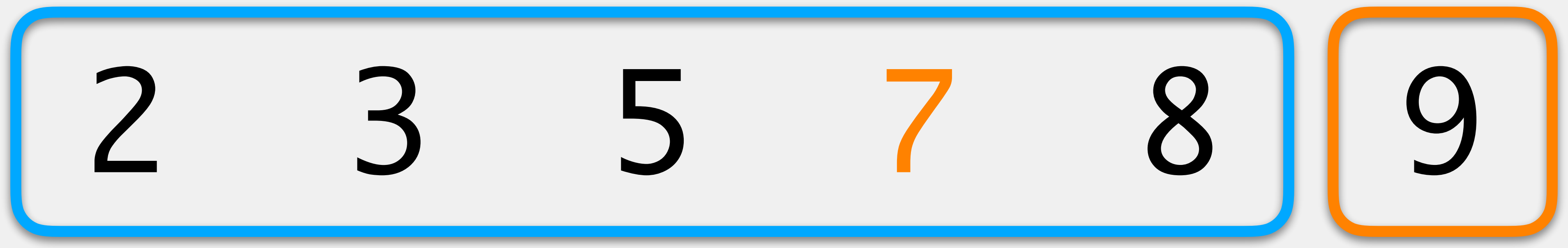
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one



2 3 5 7 8 | 9

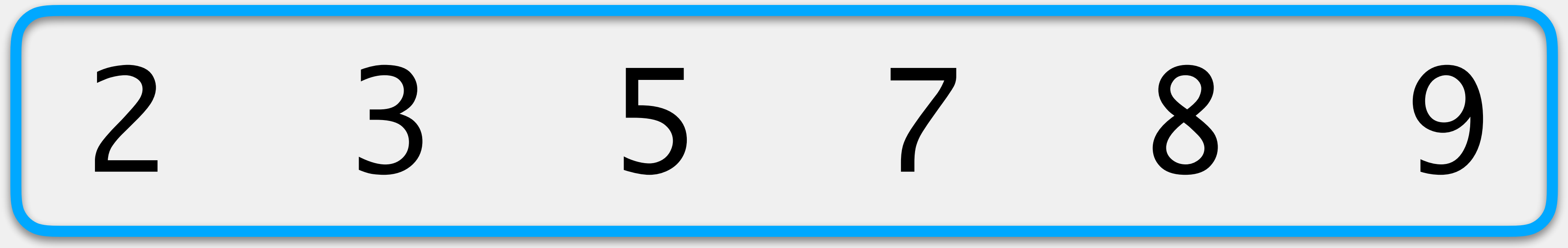
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

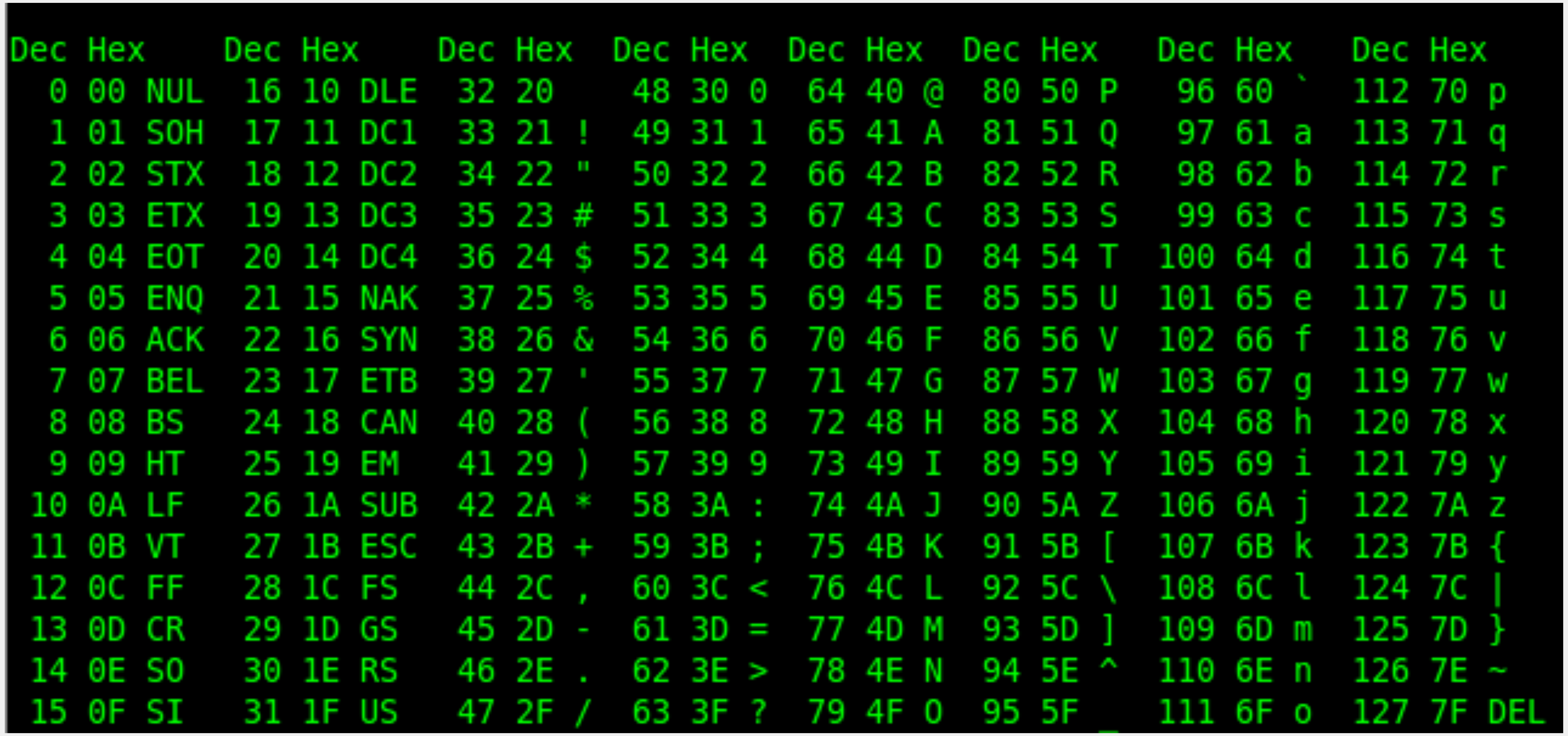


2 3 5 7 8 9

char Datatype & Sorting

chars have a strict ordering, just like numbers

- comes from underlying numeric representations every char has



```
Dec Hex     Dec Hex     Dec Hex  Dec Hex  Dec Hex  Dec Hex   Dec Hex   Dec Hex
  0 00 NUL  16 10 DLE  32 20    48 30 0  64 40 @  80 50 P   96 60 `  112 70 p
  1 01 SOH  17 11 DC1  33 21 !  49 31 1  65 41 A  81 51 Q   97 61 a  113 71 q
  2 02 STX  18 12 DC2  34 22 "  50 32 2  66 42 B  82 52 R   98 62 b  114 72 r
  3 03 ETX  19 13 DC3  35 23 #  51 33 3  67 43 C  83 53 S   99 63 c  115 73 s
  4 04 EOT  20 14 DC4  36 24 $  52 34 4  68 44 D  84 54 T  100 64 d  116 74 t
  5 05 ENQ  21 15 NAK  37 25 %  53 35 5  69 45 E  85 55 U  101 65 e  117 75 u
  6 06 ACK  22 16 SYN  38 26 &  54 36 6  70 46 F  86 56 V  102 66 f  118 76 v
  7 07 BEL  23 17 ETB  39 27 '  55 37 7  71 47 G  87 57 W  103 67 g  119 77 w
  8 08 BS   24 18 CAN  40 28 (  56 38 8  72 48 H  88 58 X  104 68 h  120 78 x
  9 09 HT   25 19 EM   41 29 )  57 39 9  73 49 I  89 59 Y  105 69 i  121 79 y
 10 0A LF   26 1A SUB  42 2A *  58 3A :  74 4A J  90 5A Z  106 6A j  122 7A z
 11 0B VT   27 1B ESC  43 2B +  59 3B ;  75 4B K  91 5B [  107 6B k  123 7B {
 12 0C FF   28 1C FS   44 2C ,  60 3C <  76 4C L  92 5C \  108 6C l  124 7C |
 13 0D CR   29 1D GS   45 2D -  61 3D =  77 4D M  93 5D ]  109 6D m  125 7D }
 14 0E SO   30 1E RS   46 2E .  62 3E >  78 4E N  94 5E ^  110 6E n  126 7E ~
 15 0F SI   31 1F US   47 2F /  63 3F ?  79 4F O  95 5F _  111 6F o  127 7F DEL
```