

# CSCI 132:

# Basic Data Structures and Algorithms

Sorting (Bubble Sort)

Reese Pearsall  
Spring 2025

# Sorting

We will spend the next several lectures discussing how to **sort** a set of values (typically an Array of ints)

Sorting a dataset is a very frequently done task, and working with a sorted dataset is much easier than working with an unsorted dataset

Instead of saying `Array.Sort()`, we will write **four** different sorting algorithms

# Sorting

We will spend the next several lectures discussing how to **sort** a set of values (typically an Array of ints)

Sorting a dataset is a very frequently done task, and working with a sorted dataset is much easier than working with an unsorted dataset

Instead of saying `Array.Sort()`, we will write **four** different sorting algorithms

*First, let's write a method that will generate an N-sized array filled with random integers (1-100)*

# Sorting

```
public int[] getRandomArray(int n) {  
    int[] array = new int[n];  
    Random rand = new Random();  
    for(int i = 0; i < array.length; i++) {  
        array[i] = rand.nextInt(101);  
    }  
    return array;  
}
```

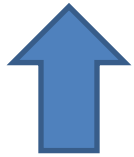
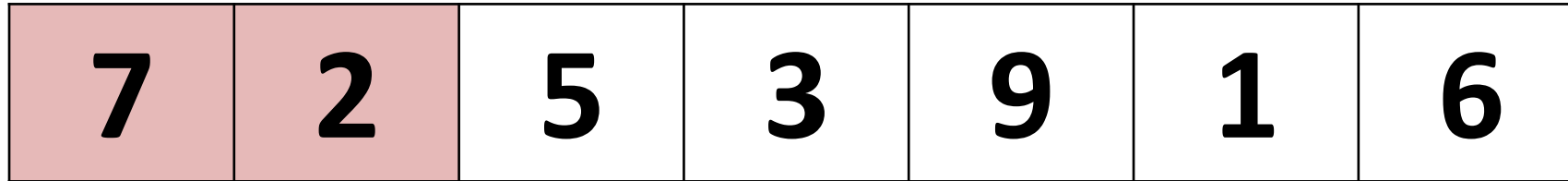
# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted

7	2	5	3	9	1	6
---	---	---	---	---	---	---

# Bubble Sort

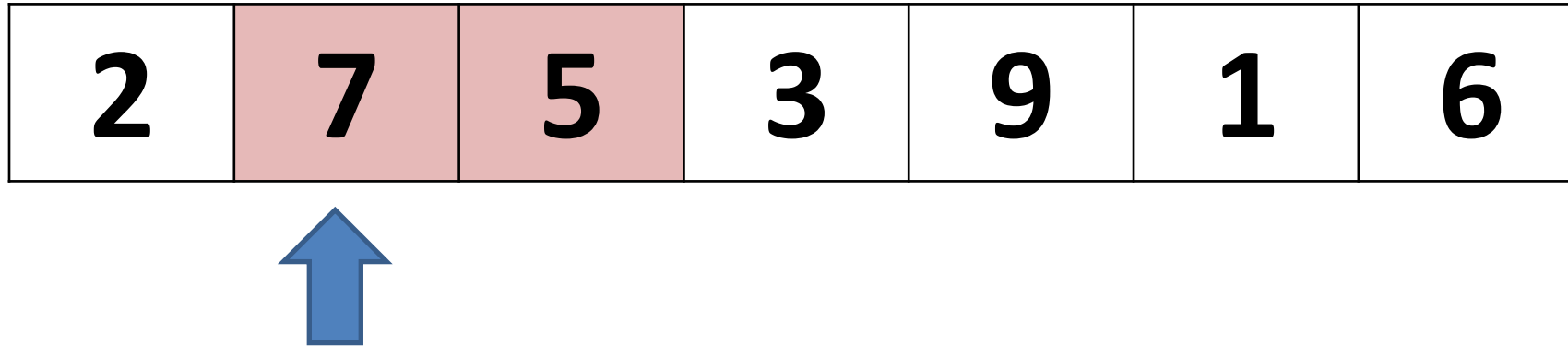
Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



Is 7 greater than 2? → SWAP

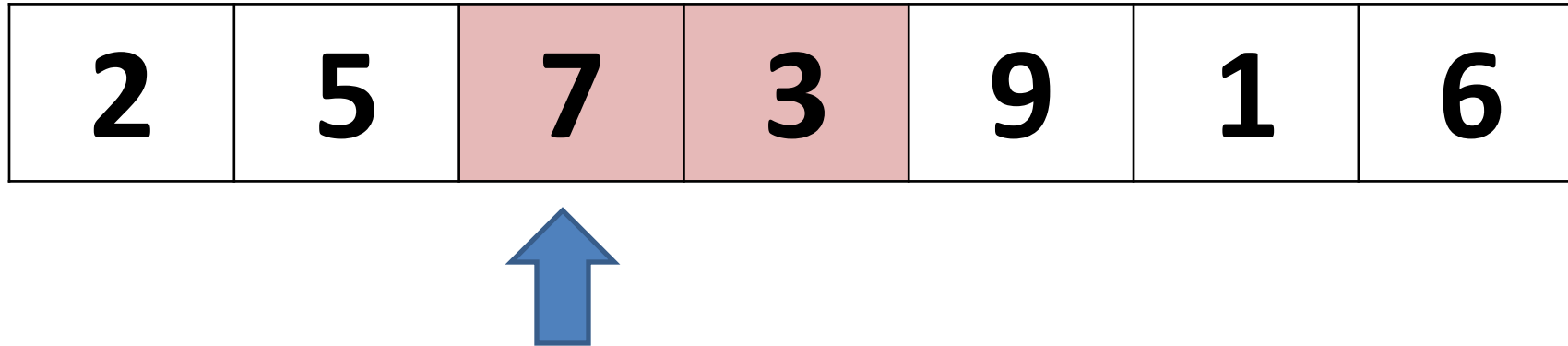
# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



# Bubble Sort

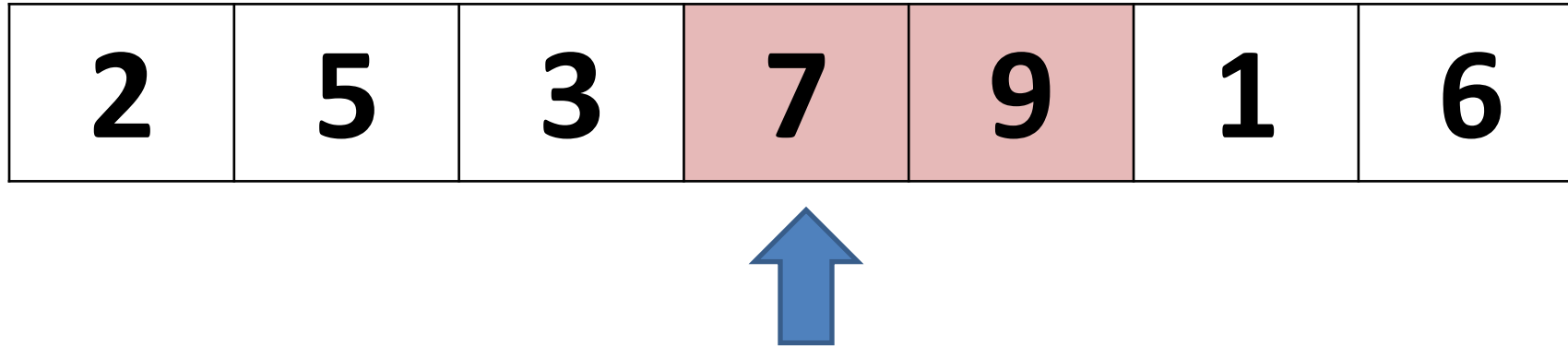
Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted





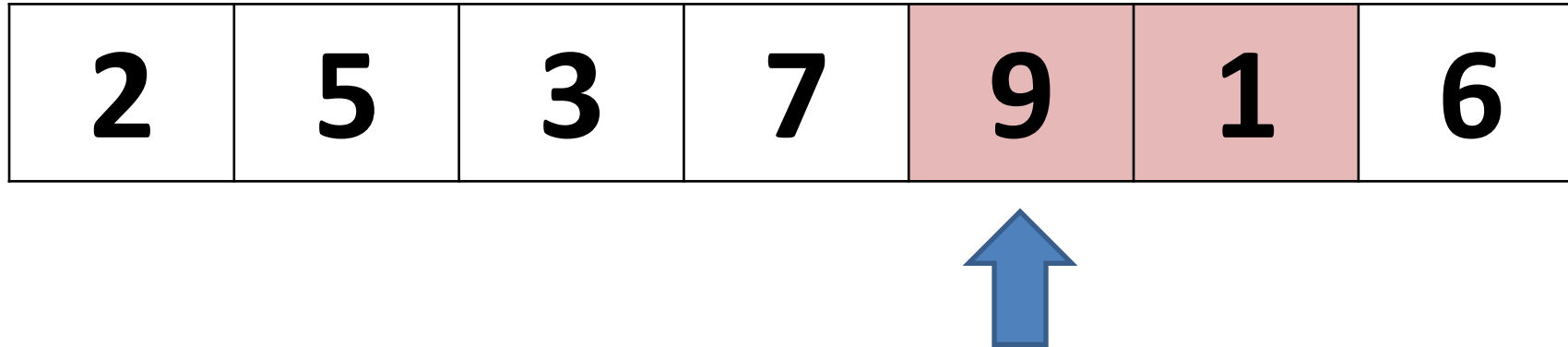
# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



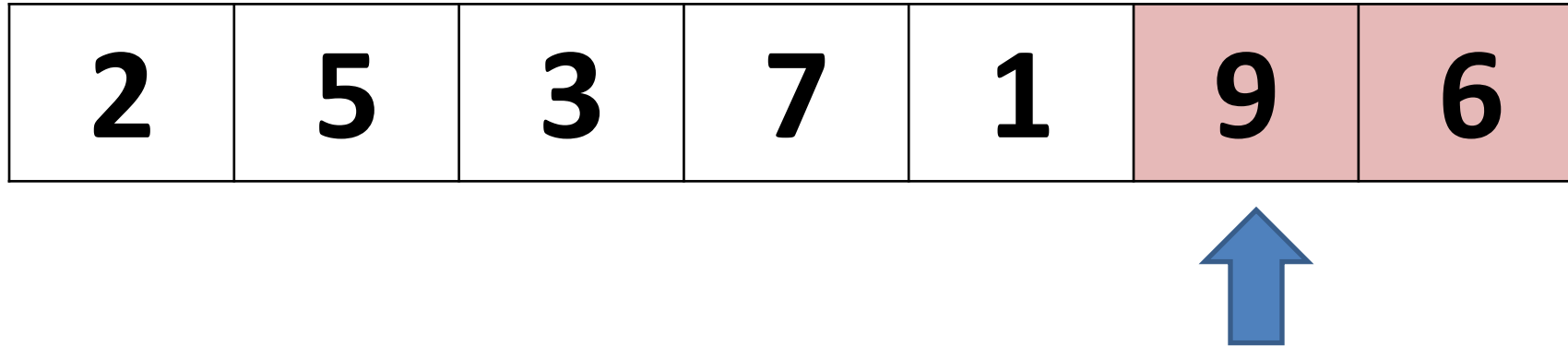
# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



# Bubble Sort

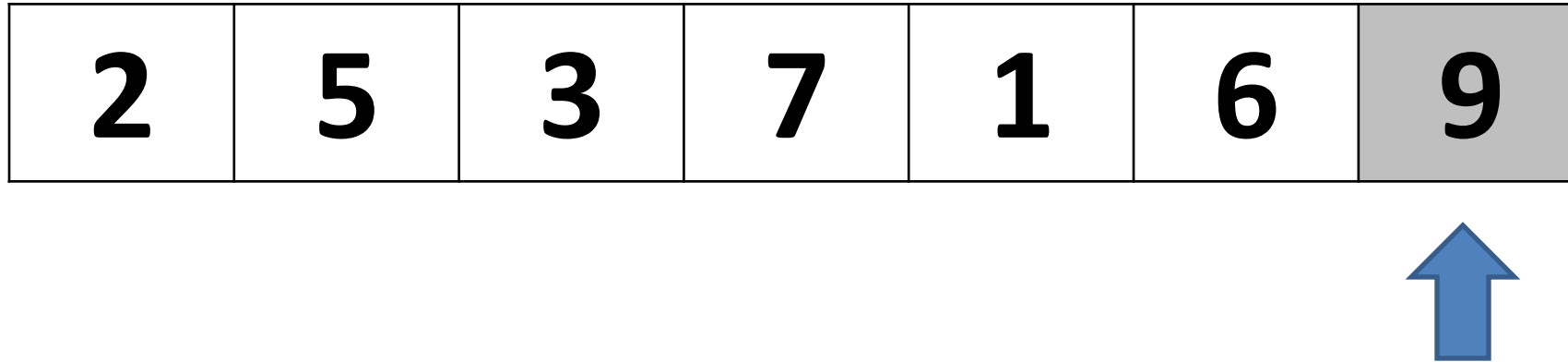
Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted

2	5	3	7	1	6	9
---	---	---	---	---	---	---



# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted

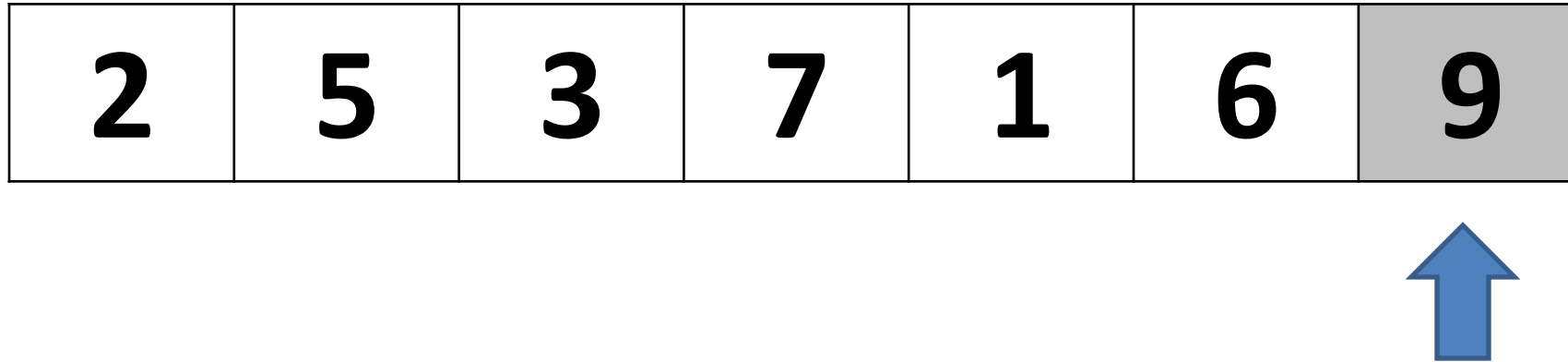


At this point, 9 (the biggest number) is at the correct spot in the array.

So, we no longer need to check that index!

# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



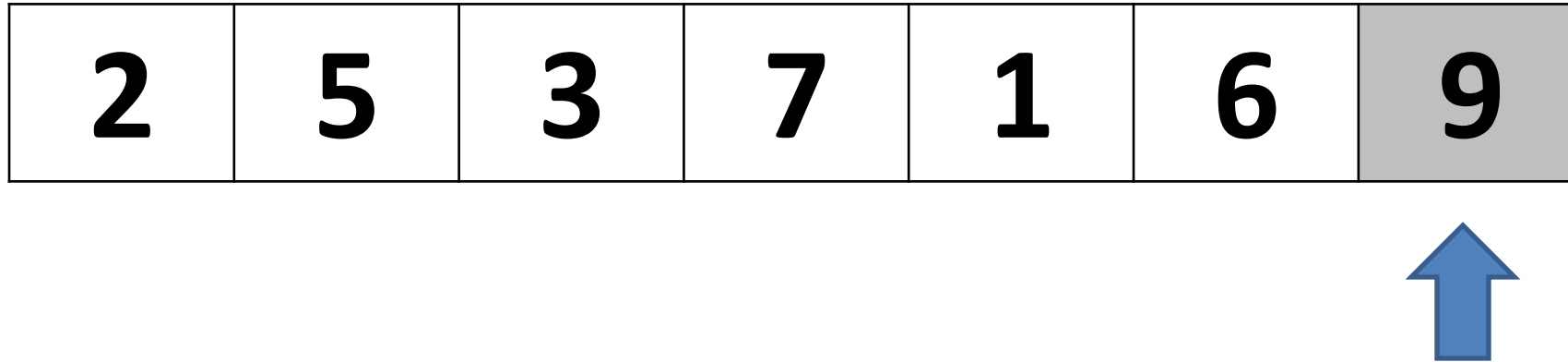
At this point, 9 (the biggest number) is at the correct spot in the array.

So, we no longer need to check that index!

Bubble Sort → “The biggest bubbles rise to the top naturally”

# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



At this point, 9 (the biggest number) is at the correct spot in the array.

So, we no longer need to check that index!

Now, we start over again, but now we check one less spot!

# Bubble Sort

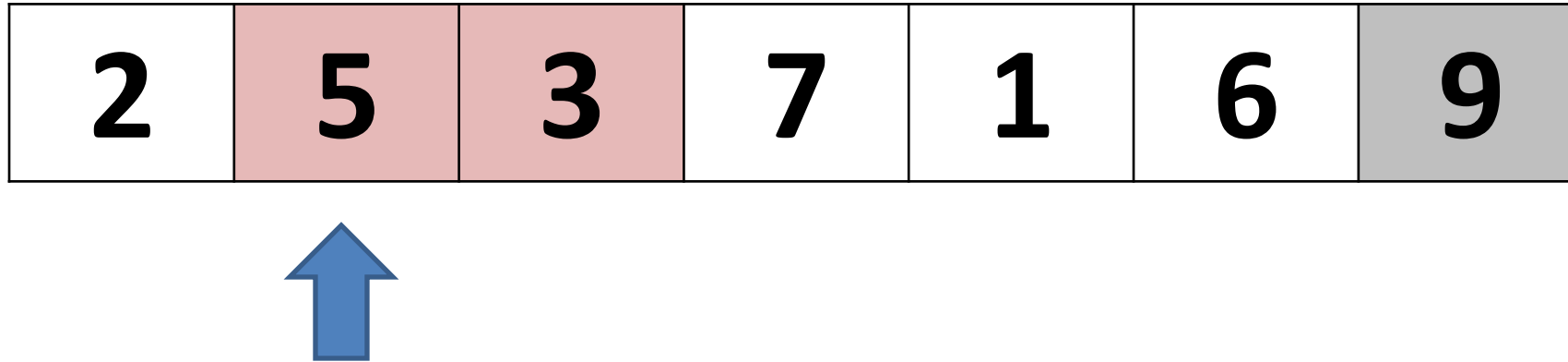
Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted





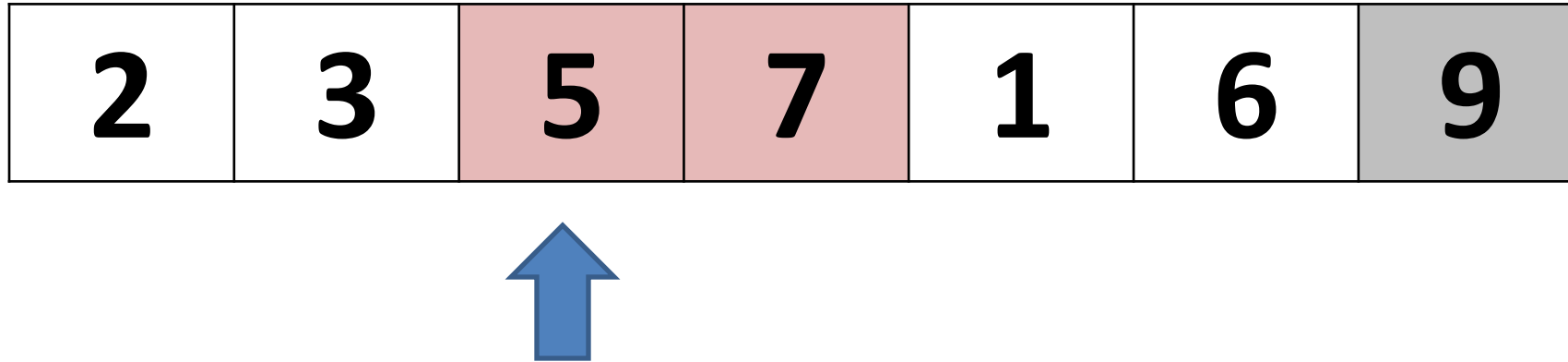
# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



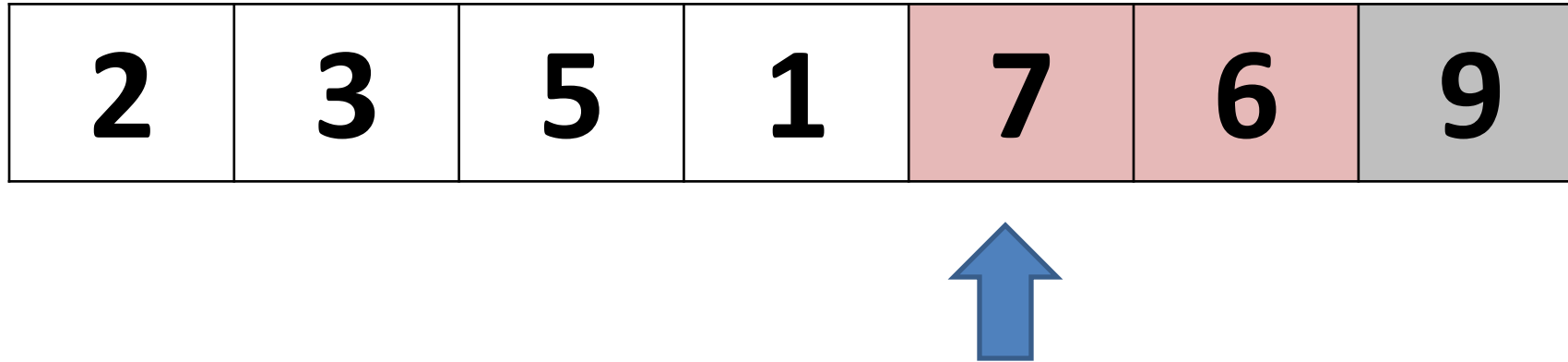
# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



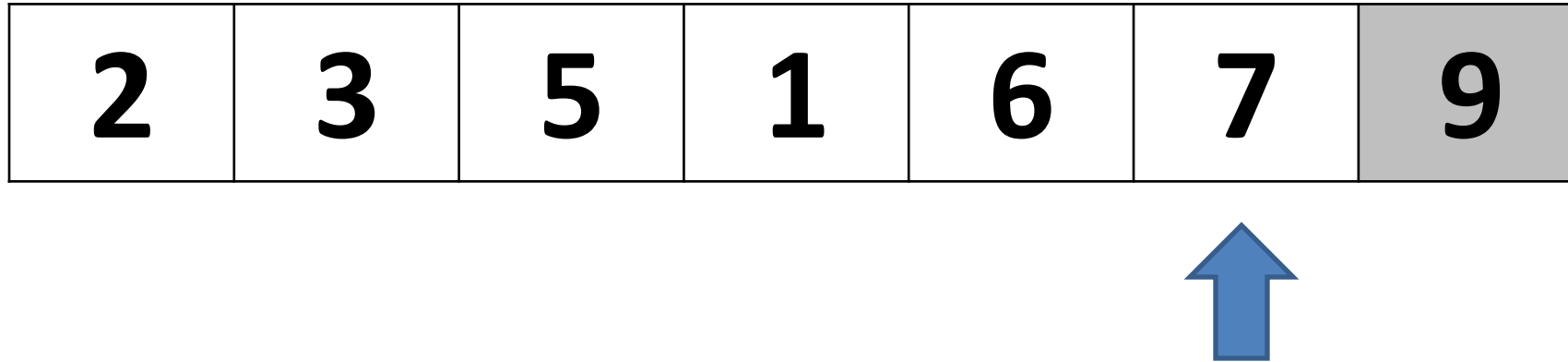
# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



7 is now in the correct spot of the array

# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted

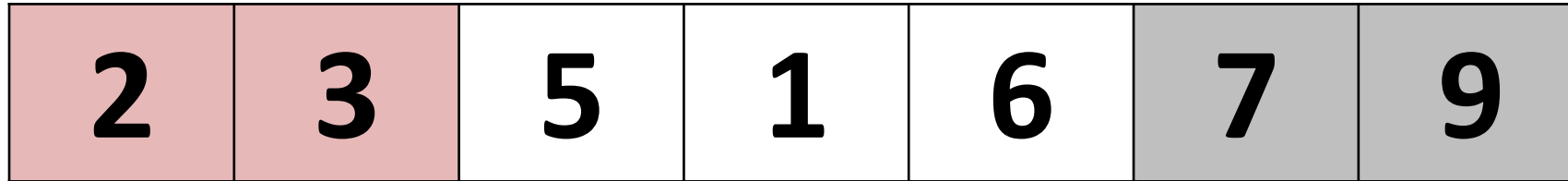
2	3	5	1	6	7	9
---	---	---	---	---	---	---



Repeat again!

# Bubble Sort

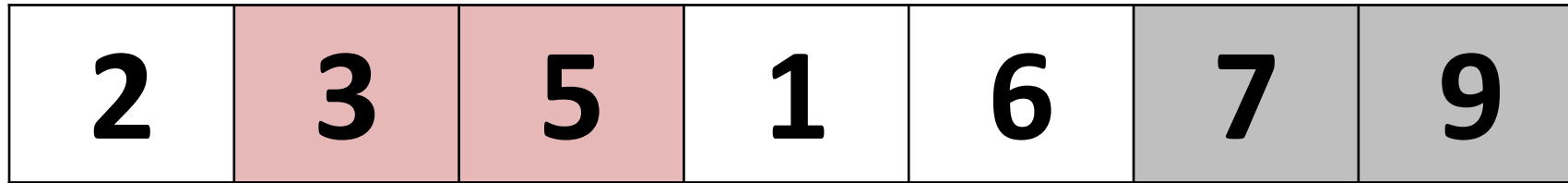
Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



Repeat again!

# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted

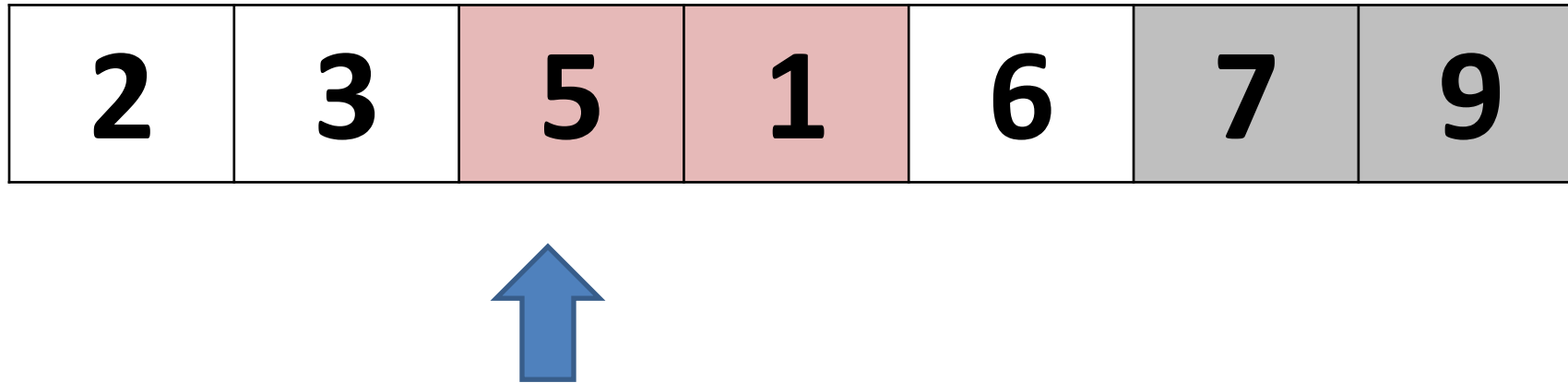


Repeat again!



# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



Repeat again!

# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted



Repeat again!

# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted

2	3	1	5	6	7	9
---	---	---	---	---	---	---



Repeat again!

# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted

1	2	3	5	6	7	9
---	---	---	---	---	---	---



(fast forwarding....)

# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted

1	2	3	5	6	7	9
---	---	---	---	---	---	---

All done!

# Bubble Sort

Iterate through an array and compare **pairs** in the array. When comparing two numbers, if one is bigger than the other, **swap**. Keep iterating until array is sorted

1	2	3	5	6	7	9
---	---	---	---	---	---	---

All done!

# Bubble Sort

```
public void bubbleSort(int[] array) {  
    int n = array.length;  
    for(int i = 0; i < n - 1; i++) {  
        for(int j = 0; j < n - i - 1; j++) {  
            if( array[j] > array[j + 1]) {  
                //swap  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j + 1] = temp;  
            }  
        }  
    }  
}
```

# Bubble Sort

```
public void bubbleSort(int[] array) {  
    int n = array.length;  
    for(int i = 0; i < n - 1; i++) {  
        for(int j = 0; j < n - i - 1; j++) {  
            if( array[j] > array[j + 1]) {  
                //swap  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j + 1] = temp;  
            }  
        }  
    }  
}
```

Running time?



# Bubble Sort

```
public void bubbleSort(int[] array) {  
    int n = array.length; O(1)  
    for(int i = 0; i < n - 1; i++) { O(n)  
        for(int j = 0; j < n - i - 1; j++) { O(n)  
            if( array[j] > array[j + 1]) { O(1)  
                //swap  
                int temp = array[j]; O(1)  
                array[j] = array[j+1]; O(1)  
                array[j + 1] = temp; O(1)  
            }  
        }  
    }  
}
```

Running time?

# Bubble Sort

```
public void bubbleSort(int[] array) {  
    int n = array.length; O(1)  
    for(int i = 0; i < n - 1; i++) { O(n)  
        for(int j = 0; j < n - i - 1; j++) { O(n)  
            if( array[j] > array[j + 1]) { O(1)  
                //swap  
                int temp = array[j]; O(1)  
                array[j] = array[j+1]; O(1)  
                array[j + 1] = temp; O(1)  
            }  
        }  
    }  
}
```

Running time?  **$O(n^2)$**

$n = | \text{array} |$

For loop in a for loop =  $n * n$

# Bubble Sort

Bubble Sort Gif

<https://upload.wikimedia.org/wikipedia/commons/c/c8/Bubble-sort-example-300px.gif>

# Bubble Sort (Recursion)

## **Base Case:**

If we have one array element left to sort, return

## **Recursive case:**

Do one loop of bubble sort, call the method again and pass the array except for the last element (last element is already in place!)

# Bubble Sort (Recursion)

```
public void bubbleSortRecursion(int[] array, int n) {  
    if (n == 1) {  
        return;  
    }  
    for(int i = 0; i < n - 1; i++) {  
        if(array[i] > array[i+1]) {  
            int temp = array[i];  
            array[i] = array[i+1];  
            array[i+1] = temp;  
        }  
    }  
    bubbleSortRecursion(array, n-1);  
}
```

n = size of unsorted section of array

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

7	2	5	3	9	1	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

`minimum_so_far = 7`

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

7	2	5	3	9	1	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

`minimum_so_far = 7`

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

7	2	5	3	9	1	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

`minimum_so_far = 2`



# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

7	2	5	3	9	1	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

`minimum_so_far = 2`

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

7	2	5	3	9	1	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

`minimum_so_far = 2`

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

7	2	5	3	9	1	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

`minimum_so_far = 2`

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

7	2	5	3	9	1	6
---	---	---	---	---	---	---



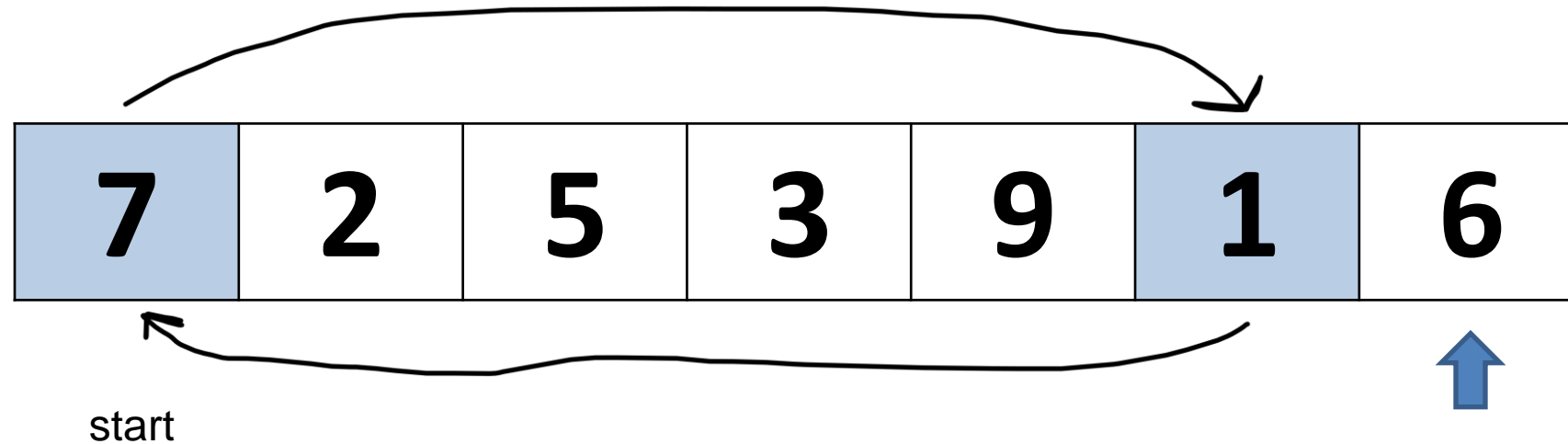
start

Goal: Find the minimum element

`minimum_so_far = 1`

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot



Goal: Find the minimum element

`minimum_so_far = 1`

Now that we've found the minimum value, swap it with the spot that we started at

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

1	2	5	3	9	7	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

`minimum_so_far = 2`

1 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

1	2	5	3	9	7	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

`minimum_so_far = 2`

1 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

1	2	5	3	9	7	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

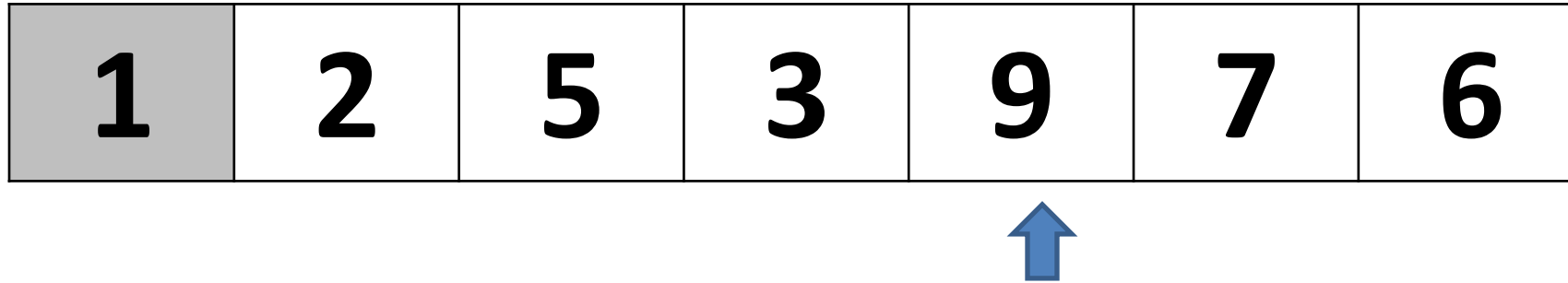
`minimum_so_far = 2`

1 is at the correct spot in the array now, so now we find the next lowest number



# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot



start

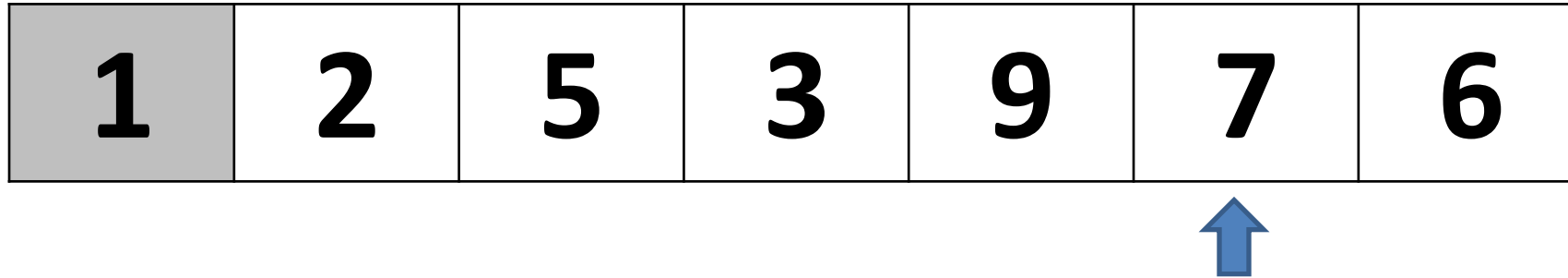
Goal: Find the minimum element

`minimum_so_far = 2`

1 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot



start

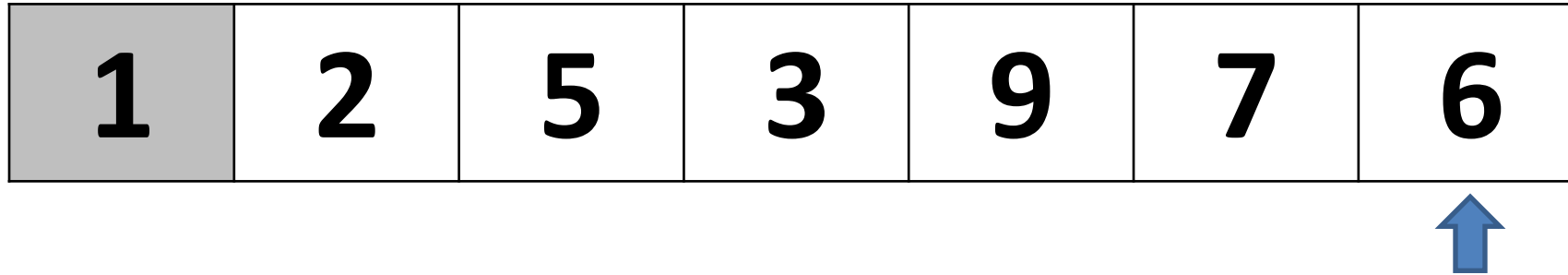
Goal: Find the minimum element

`minimum_so_far = 2`

1 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot



start

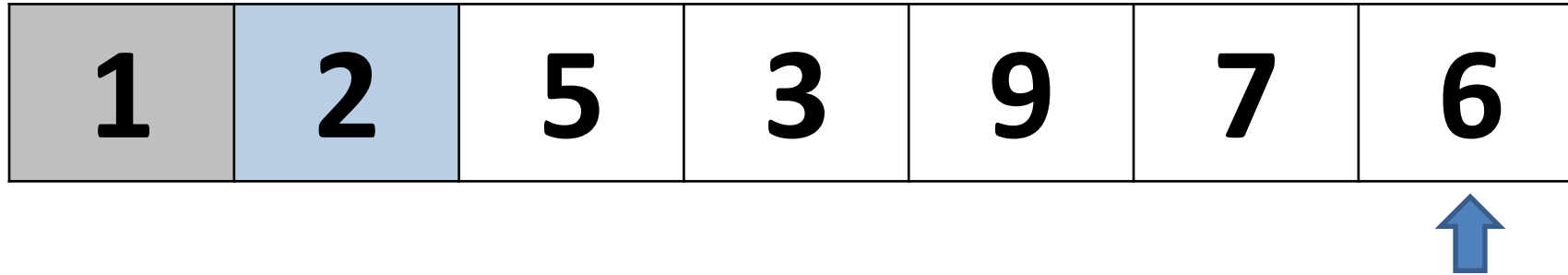
Goal: Find the minimum element

`minimum_so_far = 2`

1 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot



start

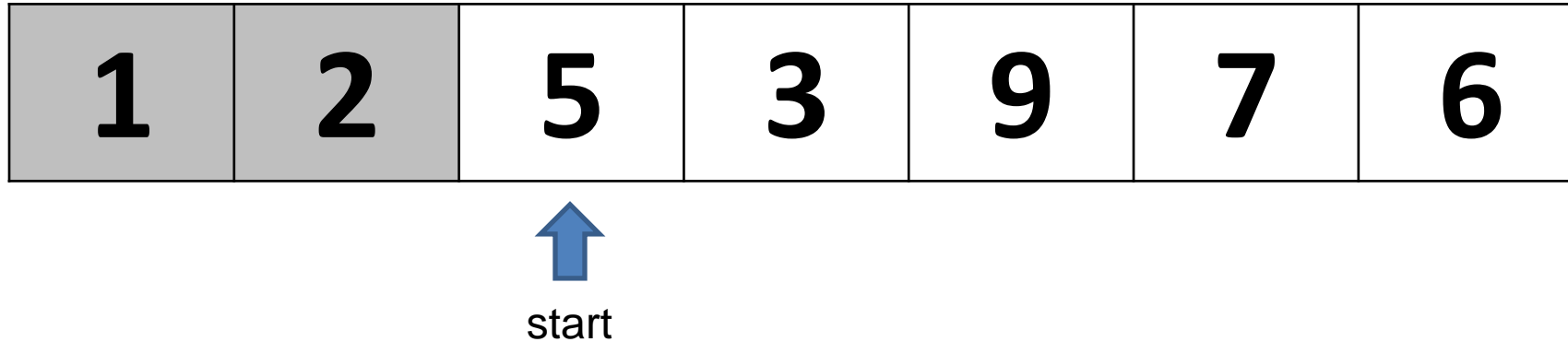
Goal: Find the minimum element

`minimum_so_far = 2`

1 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot



Goal: Find the minimum element

`minimum_so_far = 5`

2 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

1	2	5	3	9	7	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

`minimum_so_far = 5`

2 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

1	2	5	3	9	7	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

`minimum_so_far = 3`

2 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

1	2	5	3	9	7	6
---	---	---	---	---	---	---



start

Goal: Find the minimum element

`minimum_so_far = 3`

2 is at the correct spot in the array now, so now we find the next lowest number



# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot

1	2	5	3	9	7	6
---	---	---	---	---	---	---

start



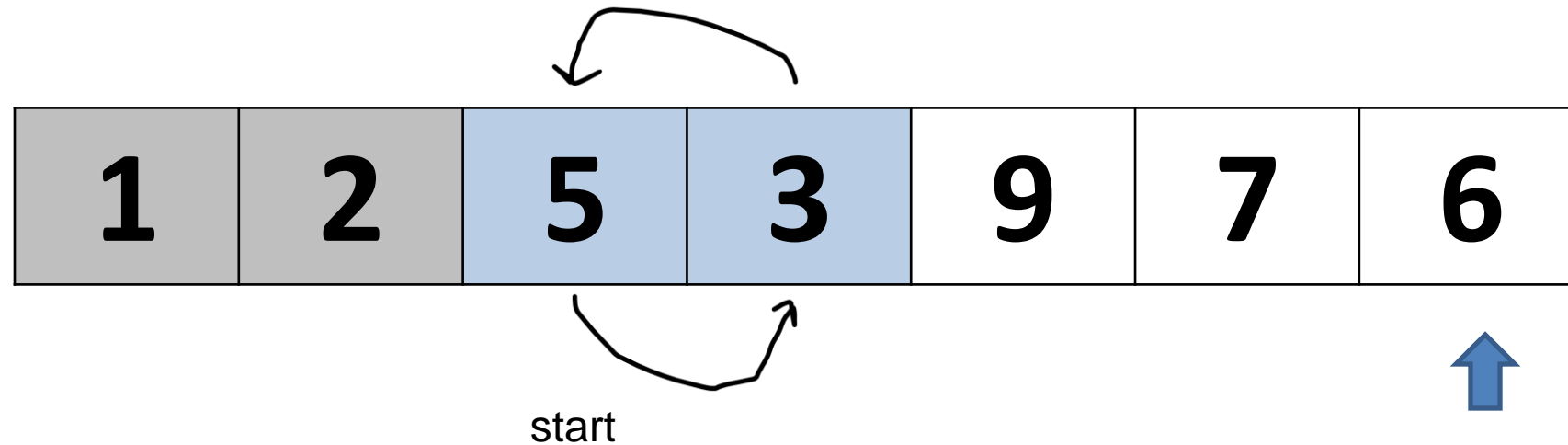
Goal: Find the minimum element

`minimum_so_far = 3`

2 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot



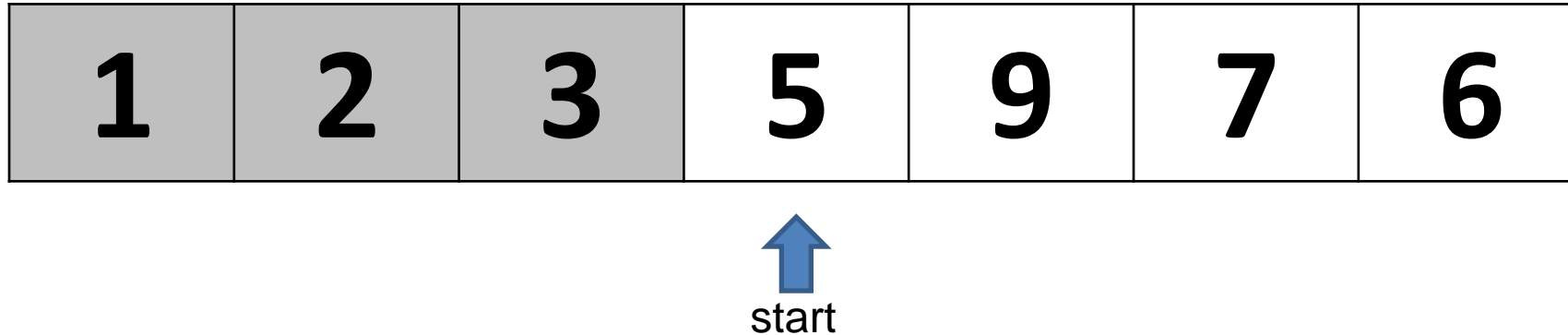
Goal: Find the minimum element

`minimum_so_far = 3`

2 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot



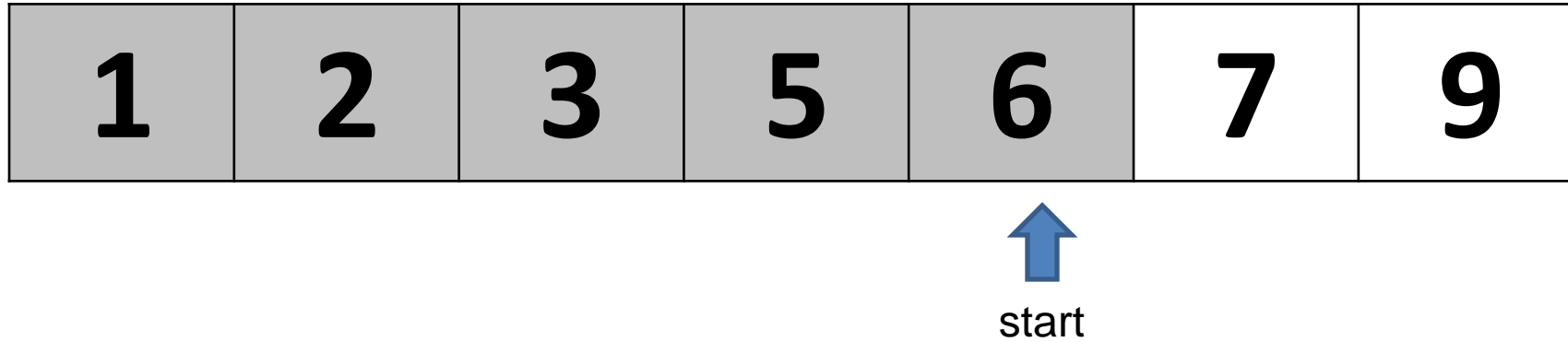
Goal: Find the minimum element

`minimum_so_far = 3`

2 is at the correct spot in the array now, so now we find the next lowest number

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot



# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot



↑  
start

# Selection Sort

Iterate through the array N times, and during each iteration, **find the minimum element** and place it in the correct spot



start

# Selection Sort

Selection Sort Gif

<https://upload.wikimedia.org/wikipedia/commons/9/94/Selection-Sort-Animation.gif>