UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Software Development II

Lecture 4 : Arrays, Sort and Search

Recommended Reading: Java for everyone Chapter6

From Last Week

Loops

- for
- while
- do-while

Input Errors and Exceptions

- Try-catch block
- Input errors

Debugging

This week

Arrays

- Declaration
- Indices
- Access
- Length
- Enhanced loop
- Search
- Copy
- 2D arrays

Sort and Search

- Linear search
- Binary search
- Selection sort
- Bubble sort

What is an Array?

- An array is a group of variables of the same data type and referred to by a common name.
- An array is a block of consecutive memory locations that hold values of the same data type.

Why do we need arrays?

Imagine we want to create a program that stores the student IDs of this class. We have 600 students.

- Solution 1: We create 600 variables to save them →

  Is this a good solution?

  600 lines of code

```
String student_1 = "w1234567";
String student_2 = "w1234568";
.
.
.
String student_600 = "w1235067";
```

- Solution 2: Arrays → `String[] students_ID = {"w1234567", "w1234568","..","w1235067"};`

- Arrays are used to store multiple values in a single variable.
- Each array has a type (same type as variables/values):
  - `int[] numbers = {1, 2, 3, 4};`

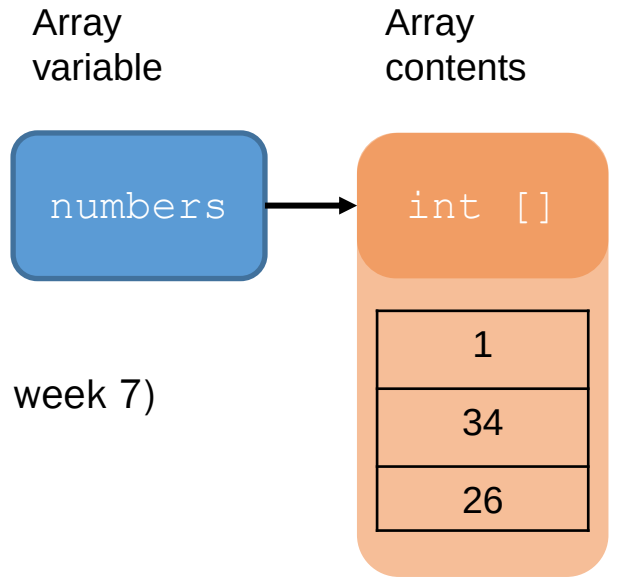
Arrays: Declaration & initialization

1. Declaration:

```
int[] numbers;
```

type — name — Array name

An array variable contains a **reference** to the array content. The **reference** is the location of the array contents (in memory)



2. Declaration and initialization with no values

```
int[] numbers = new int[size];
```

type — name — needed to create the array object (create object – week 7) — type — Length(any number)

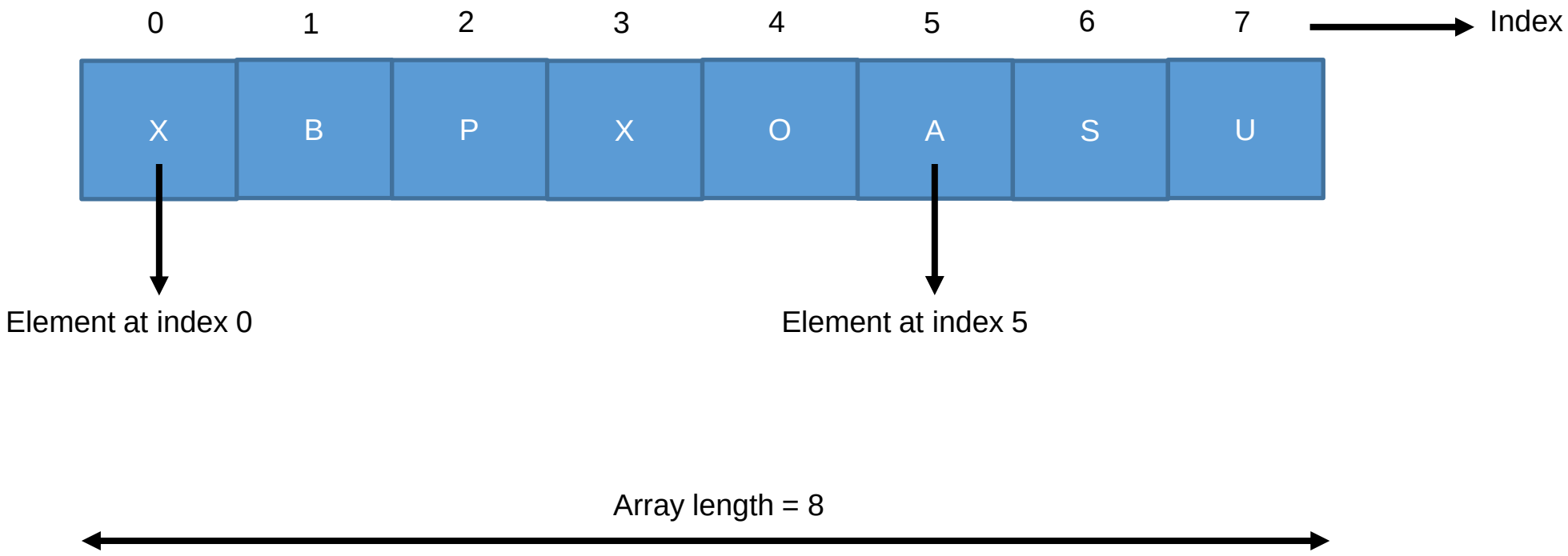
3. Declaration and initialization with values

```
int[] numbers = {1, 34, 26}
```

type — name — Values (any but must be of type, any length)

! We cannot mix different data types

Arrays: Index



- Individual locations are called array's elements.
- Each location is identified by a number, called index or subscript.
- An index can have any integer value **from 0 to array's length – 1**.

Get values from arrays

- We access the values of an array by using the index number.
- Indexes start at 0:

Index 0 Index 1 Index 2

```
String[] students_ID = {"w1234567", "w1234568", "w1234569"};
```

```
System.out.println(students_ID[0]);
```
→ Output = `w1234567`

```
System.out.println(students_ID[1]);
```
→ Output = `w1234568`

```
System.out.println(students_ID[3]);
```
→ Output = `w1234569`

Use square brackets to access the value or element

Update values from arrays

- We access the values of an array by using the index number.
- Indexes start at 0:

Index 0 Index 1 Index 2

```
String[] students_ID = {"w1234567", "w1234568", "w1234569"};
```

```
System.out.println(students_ID[0]);
```
→ Output = `w1234567`

```
Students_ID[0] = w9876543;
```

```
System.out.println(students_ID[0]);
```
→ Output = `w9876543`

How to get the Array Length?

- We can calculate the length of an array by using the ***length*** property.

Index 0 Index 1 Index 2

```
String[] students_ID = {"w1234567", "w1234568", "w1234569"};
```

```
System.out.println(students_ID.length);
```
→ Output = 3

- In Java, the length of an array is **fixed** at the time of its creation.
- Java interpreter checks the values of indices at run time and throws ArrayIndexOutOfBoundsException if an index is negative or if it is greater than the length of the array – 1.

Default initialization values

When we declare and initialize an array with no values, it will be initialized with the default value:

Type	Initialization value
int/short/byte/long	0
float/double	0.0
boolean	false
References (String, objects)	null


```
String[] stringarray = new String[10];
System.out.println(stringarray[0]);
Output : null
```

Arrays and loops

- How can we print all the values in an array?
- Which statement would you use? If/for/while?

```
String[] students_ID = {"w1234567", "w1234568", "w1234569"};

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

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

Output :

```
w9876543
w1234568
w1234569
```

Output :

```
w9876543
w1234568
w1234569
```

The enhanced loop

- You can use the enhanced *for* when you need to access every element in the array but you do not need to change any of the elements.

```
String[] students_ID = {"w1234567", "w1234568", "w1234569"};
```

```
for(int element : students_ID)
{
    System.out.println(element);
}
```

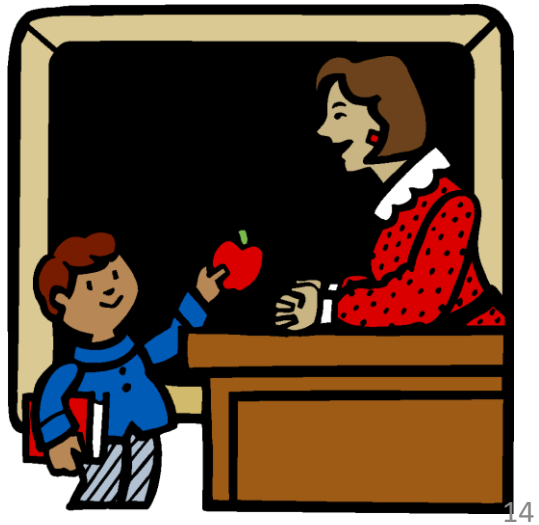
Output :

```
w9876543
w1234568
w1234569
```

Exercise: Ms. White's Test Score Analyzer

- Problem:

  Mr. White needs to analyze students' chemistry test performance. He needs a program to display a name and let him enter the test score. Then it should compute and display the average. Finally, it should give a report with names, scores, and deviation from the average.



Steps to be followed

1. Create a class to called ArrayTest, with the main method.
2. Define **`students`** array to hold names, array **`scores`** to hold test scores.
3. For each student in array
   a) display name & prompt.
   b) read double, store in array **`scores`**.
4. Compute **`overall average`**, display it.
5. For each student in array,
   a) Display name, test score, difference between that score and **`overall average`**

Output

- Display the name in a prompt
- Read the score
- Compute average
- Print summary, including deviation from mean

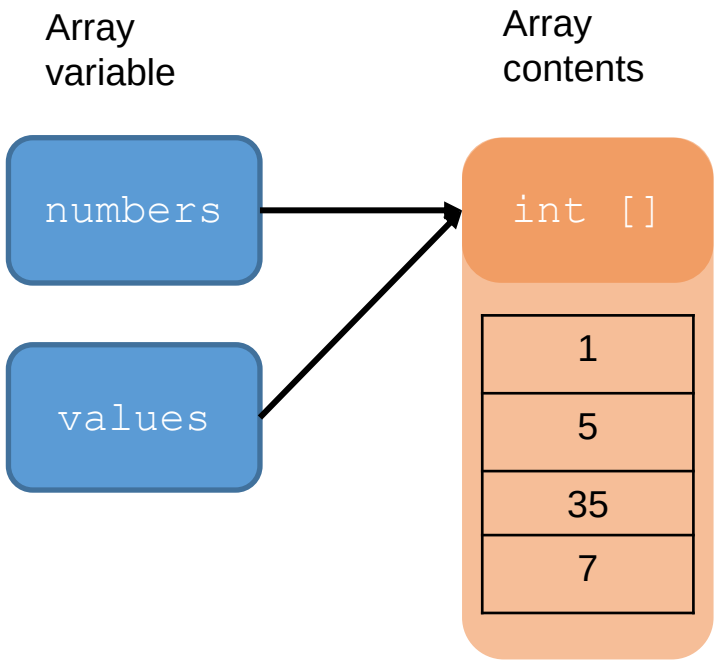
```
Test Analysis – enter scores:
Aardvark, James     : 92
Biffkirk, Sue       : 79
Crouse, Nancy       : 95
 . . .
Average = 87.39

Summary …
```

Arrays: copy

An array variable contains a **reference** to the array content.
The **reference** is the location of the array contents (in memory).

```
int[] numbers = new int [];
numbers = {1, 5, 35, 7};
```

```
int[] values = numbers;
```



Will copy the reference only, so both will point to the same memory.

What happens when we modify *values*?

Arrays: copy

- We have to be very careful when copying arrays.

```
int[] numbers = {2, 4, 6};
int[] values = numbers;

values[0] = 3;
values[1] = 5;
values[2] = 7;

for (int num : numbers){
  System.out.println(num);
}
```

We create an array

We create a copy of the variable that points to the array

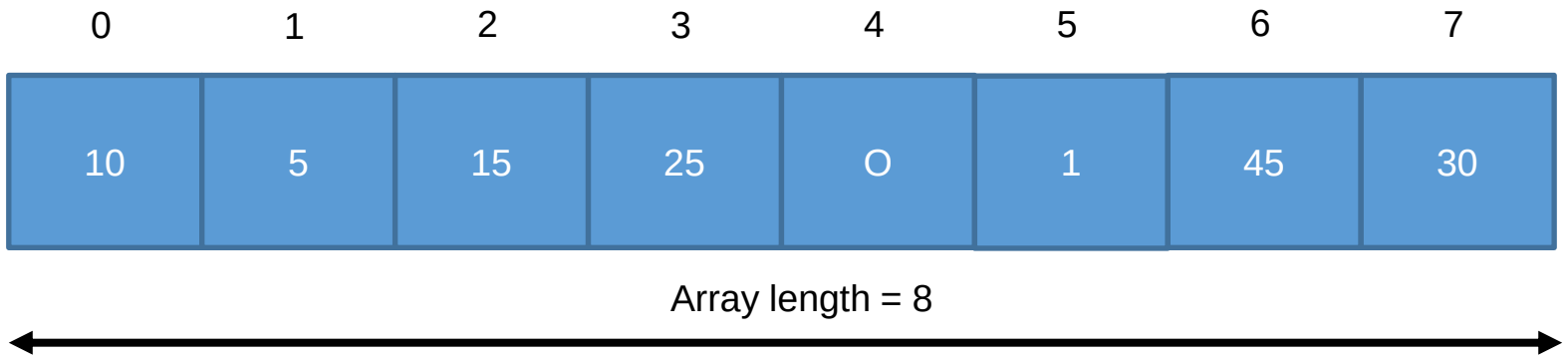
! We update the values of the copy, but we are actually also modifying the values of the original array.

The values of the original array have also changed!

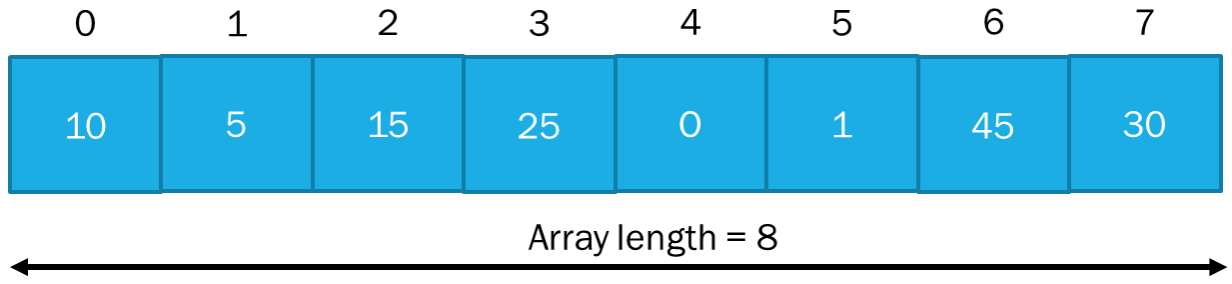
Output: 3, 5, 7.

Arrays and search

- We want to know if value 25 is in the array.
- Which statement would you use? for/while?



Arrays and search



- We want to know if value 25 is in the array.

```
int[] num_array = {10, 5, 15, 25, 0, 1, 45, 30};
int num_to_find = 25;

int i = 0;
boolean not_found = true;
while (i < num_array.length && not_found){
    if (num_array[i] == num_to_find){
        System.out.println("Found item in index: " + i);
    }
    i++;
}
```

Output :

```
Found item in index 3.
```

Algorithms: simple array sum

- Given an array of integers, we want to find the sum of its elements:

[1, 2, 3] = 1+ 2 + 3 = 6.

- Code:

```
public class sumArray {
    public static void main(String[] args) {

        int[] array = {1, 2, 3};
        int sum = 0;

        for (int i = 0; i < 3; i++)
        {
            sum = sum + array[i];
        }
        System.out.println(sum);
    }
}
```

We create the array and assign values

Initialise variable sum to 0

Loop through the array

We calculate the sum by adding the current val

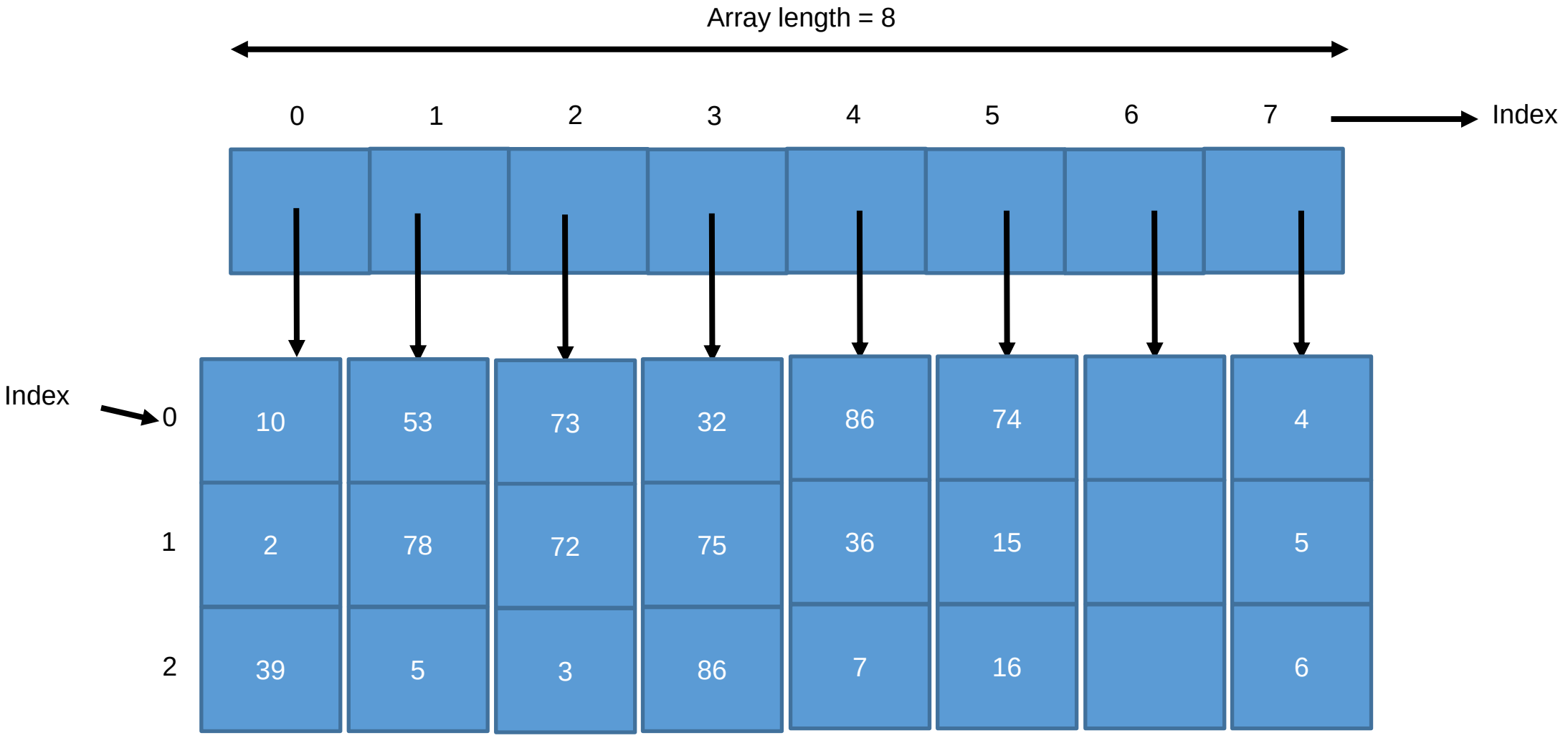
Print final result

What is the output of below program?

```
int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9};

System.out.println(array[4+2]);  //1

System.out.println(array[3]/array[1]); //2

System.out.println(array[array[4]]);  //3
```

2-DIMENSIONAL ARRAYS

Two-dimensional arrays

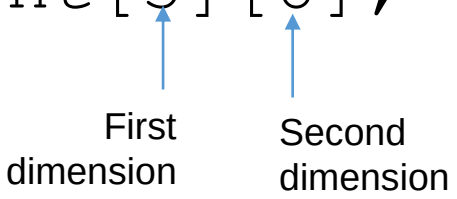


Element at index 0

Two-dimensional arrays: Declaration

- We have to define the 2 dimensions

```
int[][] numbers = new int[3][8];
```



Example:

```
int[][] numbers = {{1,2},{2,5},4,5}}
```

! We cannot mix different data types

Two-dimensional arrays: accessing values

```
int[][] numbers = new int[3][4];
numbers[1][1] = 10;
```

…

	0	1	2	3
0	numbers[0][0]	numbers[0][1]	numbers[0][2]	numbers[0][3]
1	numbers[1][0]	**10**	numbers[1][2]	numbers[1][3]
2	numbers[2][0]	numbers[2][1]	numbers[2][2]	numbers[2][3]

What is the output of below program?

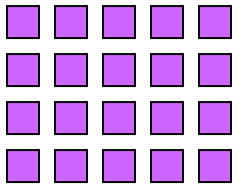
```
int[][] array = {{1,2,3},{4,5,6},{7,8,9}};
System.out.println(array[1][2]);
```

Output?

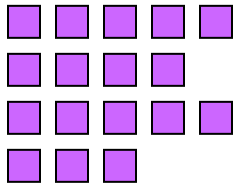
Types of 2-D Array

- Java allows “ragged” arrays, in which different rows have different lengths.

Rectangular array:



“Ragged” array:



More about 2-D Arrays

- In a two-dimensional array,

1. You need to allocate memory for only the first dimension.
2. You can allocate the remaining dimensions separately.
3. When you allocate memory to the second dimension, you can also allocate different number to each dimension.

```
int ragArr[][] = new int[3][];
ragArr[0] = new int[1];
ragArr[1] = new int[4];
```

Getting the 2-D Array Length

- In Java, a 2-D array is basically a 1-D array of 1-D arrays. Each row is stored in a separate block of consecutive memory locations.
- If **sample** is a 2-D array;
  - **sample.length** is the number of rows.
  - **sample[n-1].length** is the length of the n^{th} row.

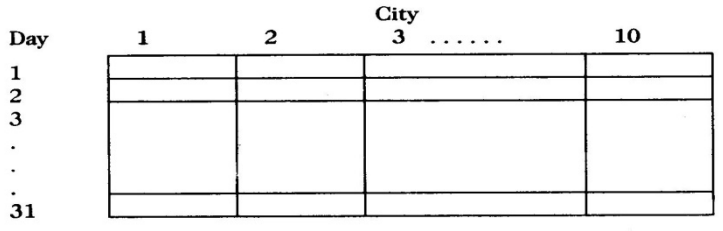
Traverse through a 2-D Array

- A 2-D array can be traversed using nested loops:

```
int [][] sample = {{ 1, 2, 3 },
                   { 4, 5, 6 }};

 for(int r=0; r<sample.length; r++) {
    for(int c=0; c<sample[r].length; c++) {
         …   //process sample[r][c]

    }
 }
```

Exercise: Weather Analyzer

- The daily maximum temperatures recorded in 10 cities during the month of January (for all 31 days)
- Write a program to read the table elements into a two-dimensional array temperature, and to find the city and day corresponding to
  - the highest temperature and
  - the lowest temperature.



SEARCH ALGORITHMS

Search

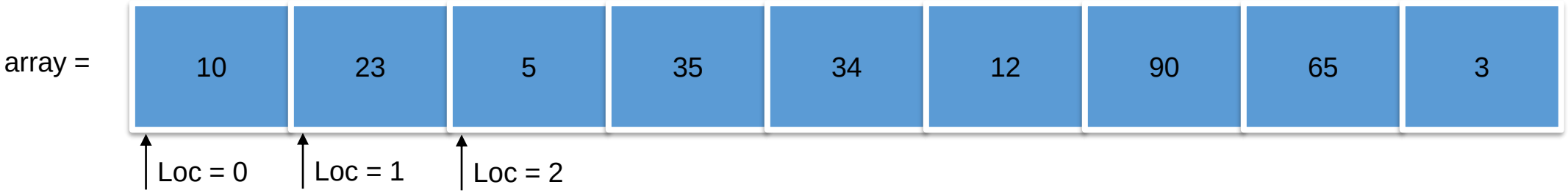
- How many F’s are in the following sentence?

“Finished files are the result of years of scientific study combined with the experience of years.”

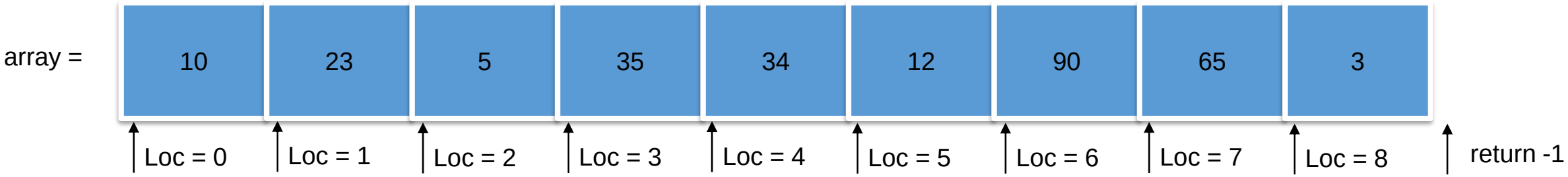
Linear Search

Given an array, we want to know if the array contains a specific value

```
LinearSearch(array, 5);
```




```
LinearSearch(array, 25);
```



Linear Search: code

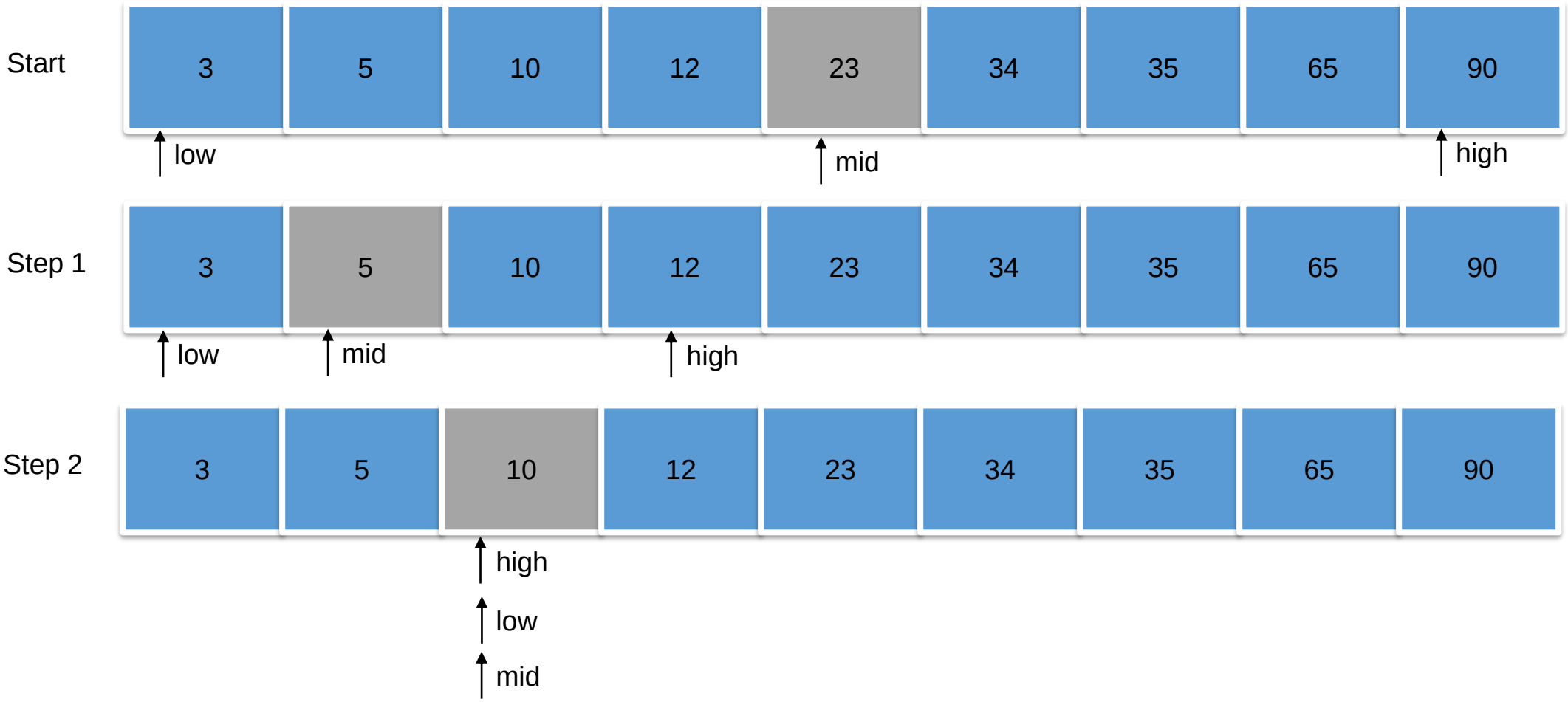
```
int[] array = {1, 12, 7, 25, 67, 46, 57};
int searchValue = 25;
int index = 0;
while (index < array.length && array[index] != searchValue) {
      index++;
}
if (index == array.length) {
      System.out.println("Value not found.");
} else {
      System.out.println("Value found in index: " + index );
}
```

Binary Search

When the array is sorted, we do not need to search the whole array .

Watch & Learn

Is the number 10 in the array?



Binary Search: code

```
int[] array = {1, 7, 12, 25, 46, 57, 67};
int searchValue = 25;
int low = 0, high = array.length - 1, mid = (low + high) / 2;
while (low <= high && array[mid] != searchValue) {
	if (array[mid] < searchValue) {
		low = mid + 1;
	} else {
		high = mid - 1;
	}
	mid = (low + high) / 2;
}
if (low > high) {
	System.out.println("Value not found.");
} else {
	System.out.println("Value found in index: " + mid);
}
```

SORT ALGORITHMS

Sorting

Sorting refers to ordering data in an increasing or decreasing manner according to some linear relationship among the data items [https://en.wikipedia.org/wiki/Sorting]

Sorting

Given an array of n values, arrange the values into ascending order.

Not sorted:

Sorted:

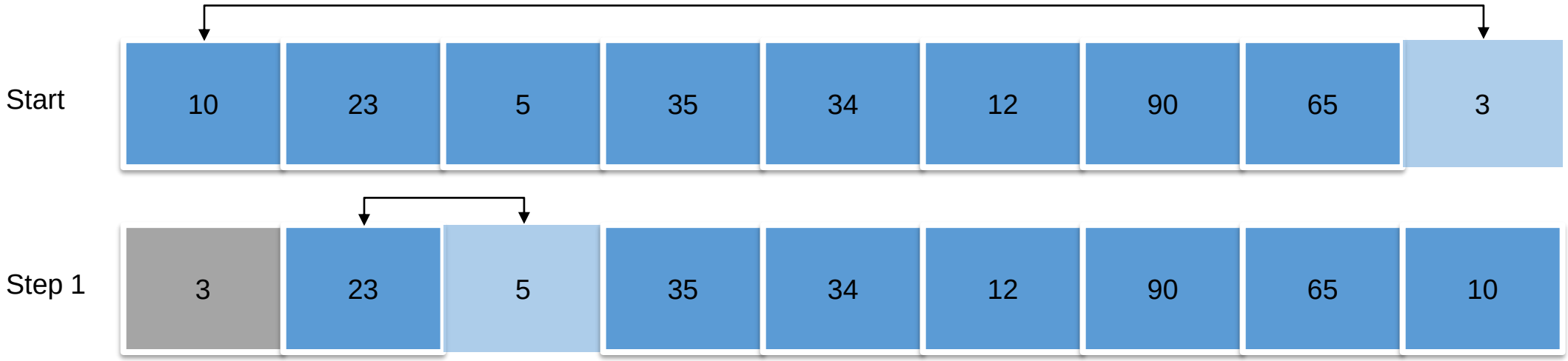
Selection sort

Selection sort sorts an array by repeatedly finding the smallest element of the unsorted tail region and moving it to the front

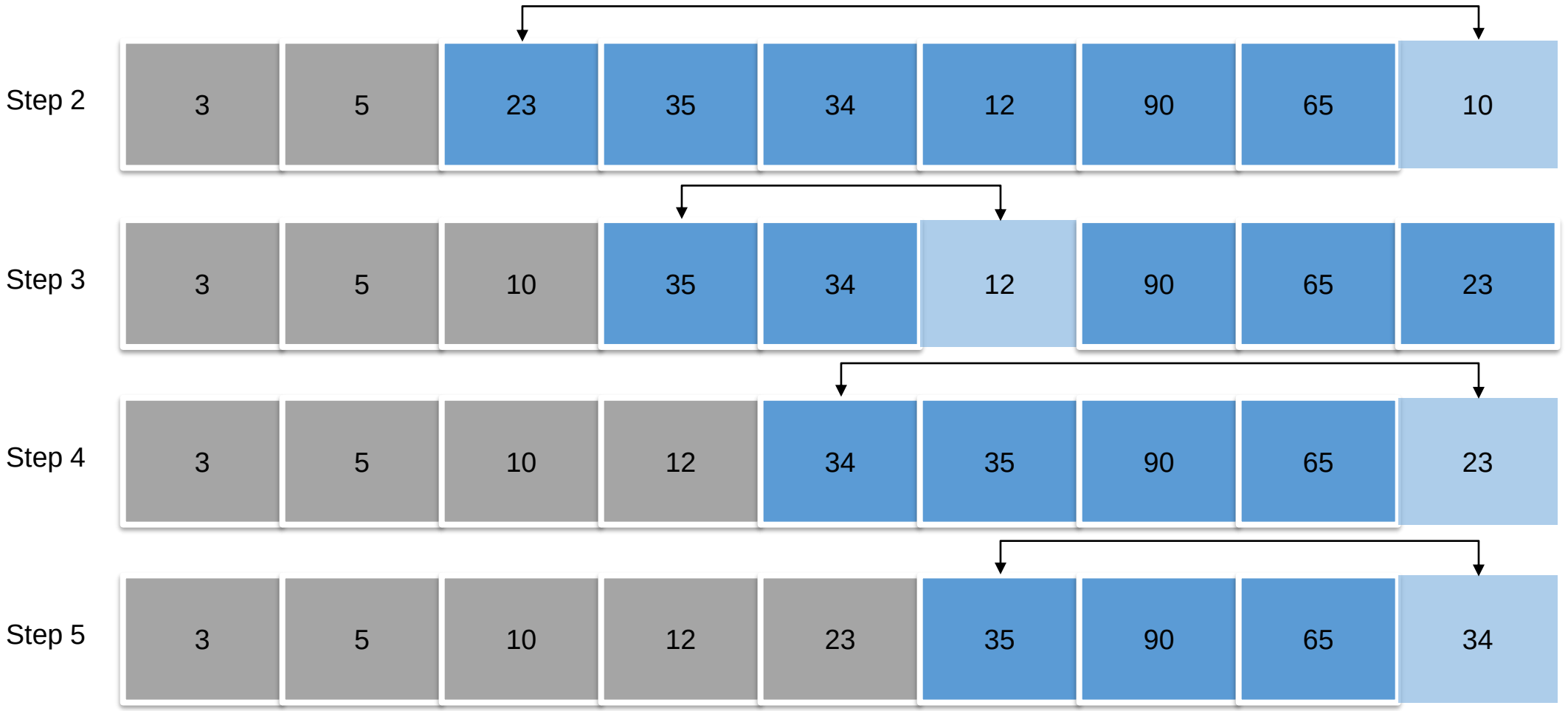
Slow on large data sets

Steps:

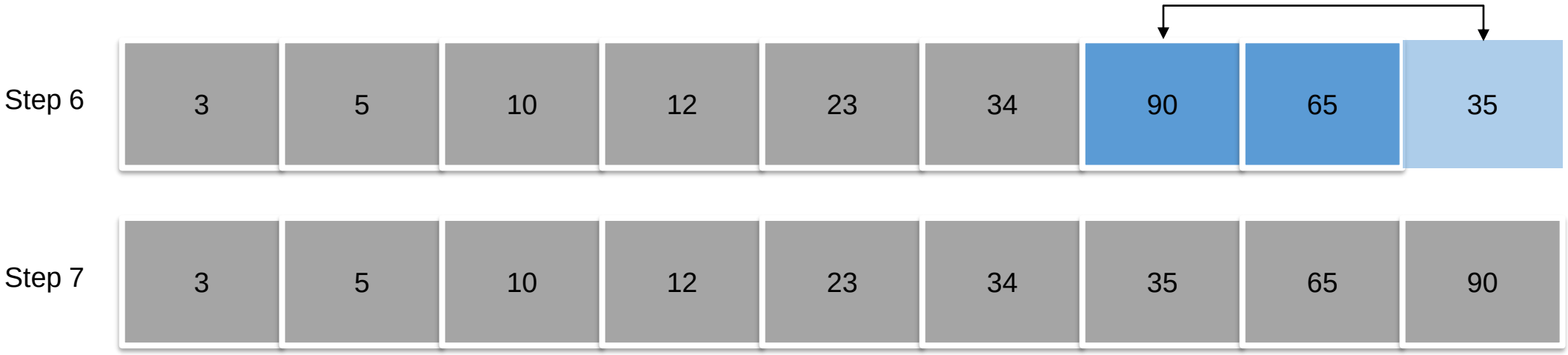
1. Find the smallest and swap it with the first element.
2. Find the smallest and swap it with the first non-sorted element.
3. Repeat step 2 until the end of the array.



Selection sort (cont.)



Selection sort (cont.)



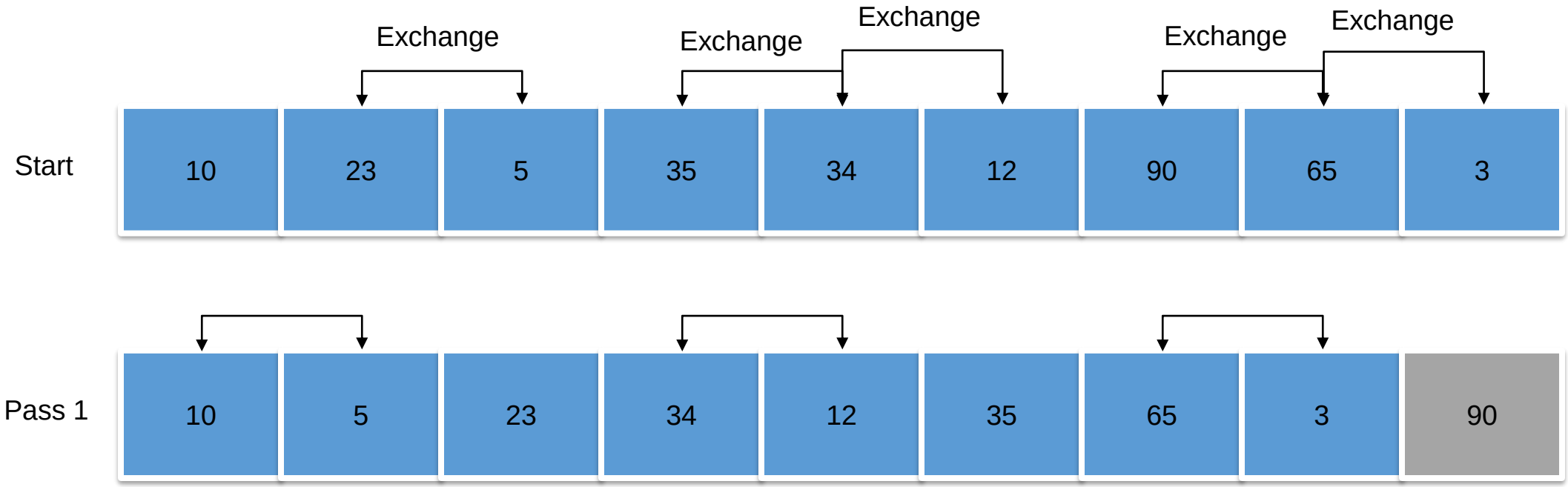
[Watch & Learn]

Selection sort: code

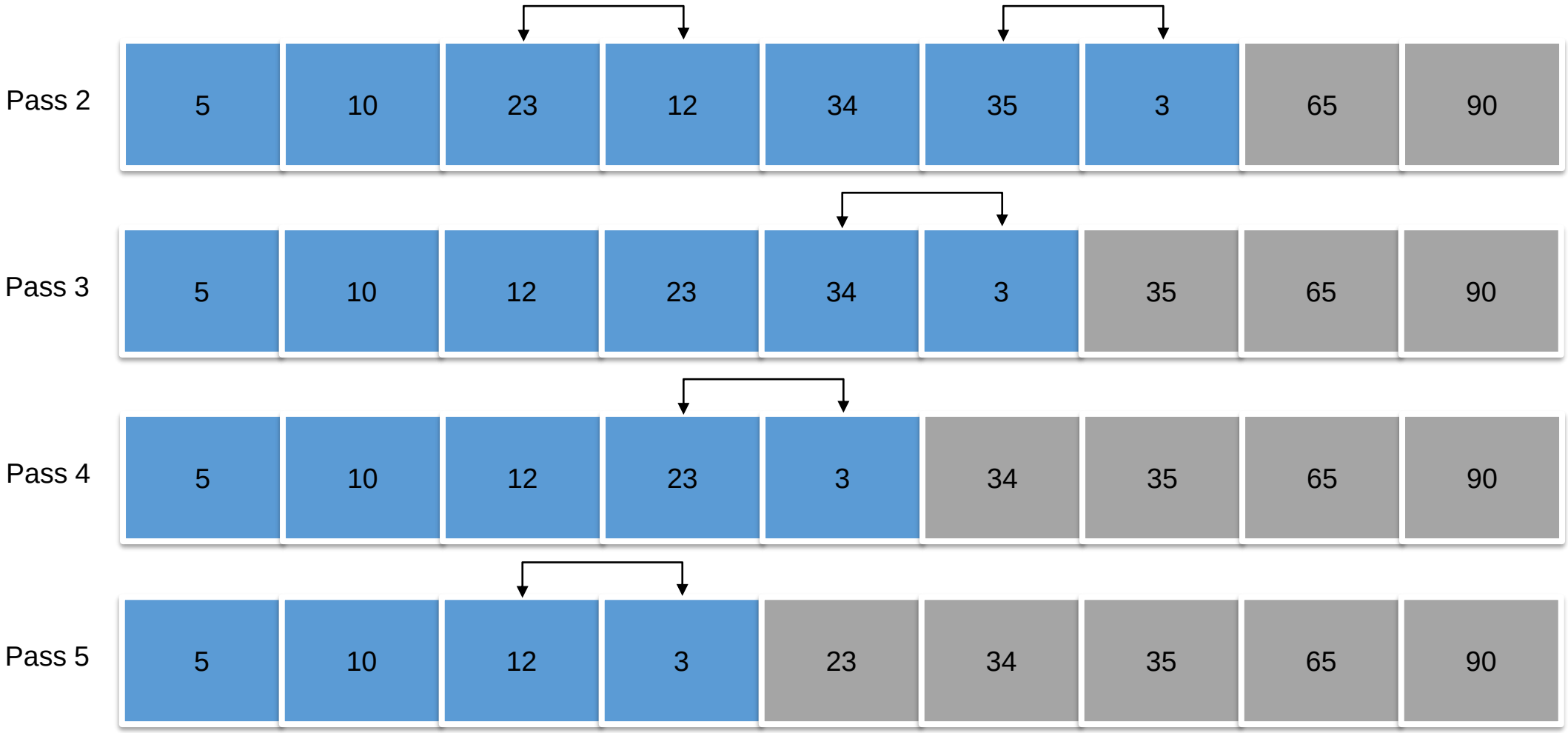
```
int[] array = {1, 12, 7, 25, 67, 46, 57};
int minIndex, temp;
for (int start = 0; start < array.length - 1; start++) {
      minIndex = start;
      for (int i = start + 1; i <= array.length - 1; i++) {
            if (array[i] < array[minIndex]) {
                  minIndex = i;
            }
      }
      temp = array[start];
      array[start] = array[minIndex];
      array[minIndex] = temp;
}
```

Bubble sort

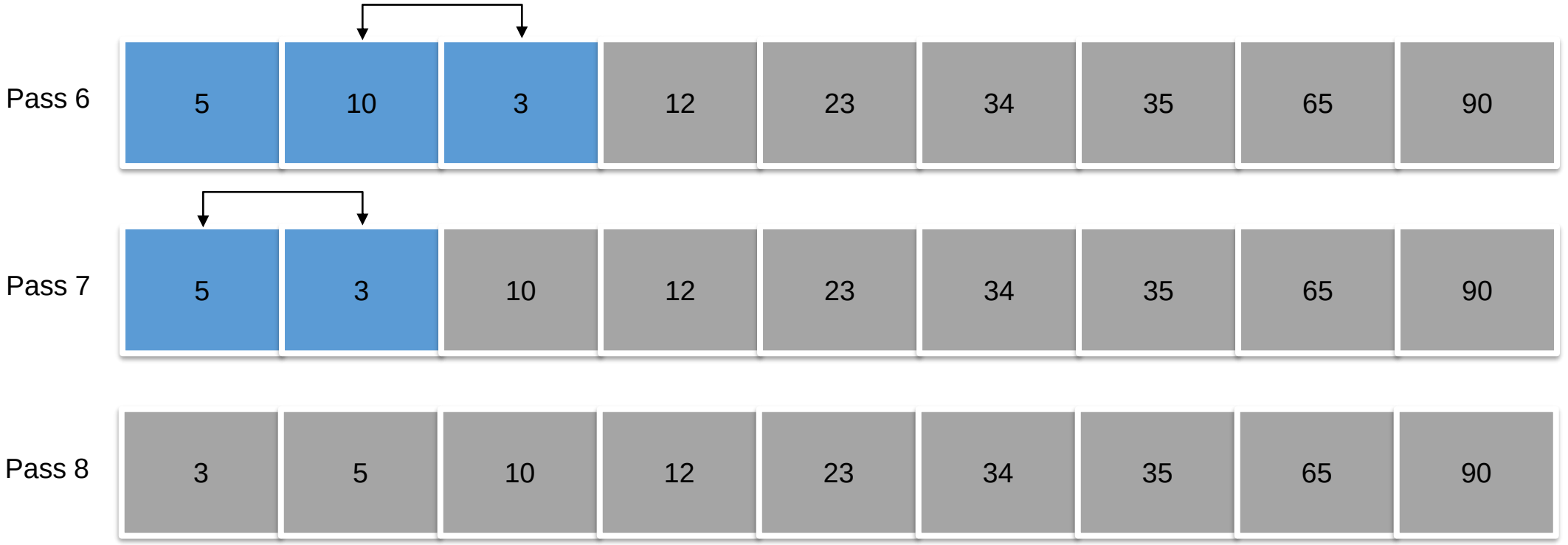
- Makes pairwise comparisons and exchange them if they are not in order.
- Steps:
  - While there are exchanges do:
    - For each element of the array, compare with the next element
    - If the current element is larger, then exchange them



Bubble sort (cont.)



Bubble sort (cont.)



Watch & Learn

Bubble sort: code

```
int[] array = {1, 12, 7, 25, 67, 46, 57};

int bottom = array.length - 2;
int temp;
boolean exchanged = true;

while (exchanged) {
        exchanged = false;
        for (int i = 0; i <= bottom; i++) {
                if (array[i] > array[i + 1]) {
                        temp = array[i];
                        array[i] = array[i + 1];
                        array[i + 1] = temp;
                        exchanged = true;
                }
        }
        bottom--;
}
```

Coursework

- Coursework specification will be available during this week on BB.
- Carefully read the coursework specification including instructions and marking scheme.
- Use Coursework Q&A padlet to ask any related questions.

Independent Study

- Complete Recommended reading *: **Java for Everyone** - Chapter 06*
- Tryout all coding examples provided in lecture slides using code editor and observe output.
- Attempt all exercises provided in lecture slides using code editor and discuss your issues during tutorials.
- Complete Formative test week 04 (Available in Blackboard Week4 folder).
- Attempt all questions in tutorial 03 and submit to BB before deadline

HackerRank questions on this topic

- Solve the following exercises in HackerRank on exception handling:
- Java 1D Array
- Java 2D Array
- Big Sorting
- Insertion Sort –Part 1 and 2
- Quicksort 1 –Partition
- Counting Sort 1 and 2


THANK
YOU