Getting To Know Algorithms: Using Arrays And Sorting

Learning Targets:

- You understand the terms *algorithms* and *data structures* and can explain these
- You can work with the elementary data structure *array*
- You know the basic *sort algorithms*

1 What Do Algorithms And Data Structures Have In Common?

The term *algorithm* is used in computer science to describe how a problem can be solved.
Maybe you have already used an algorithm in a program, for example when you had to sort a list or go through a list in order to search for an element.
You can imagine an algorithm as the mechanism which makes sure that a software program has a reliable logical flow. It therefore defines the steps one must take in order to solve a problem.

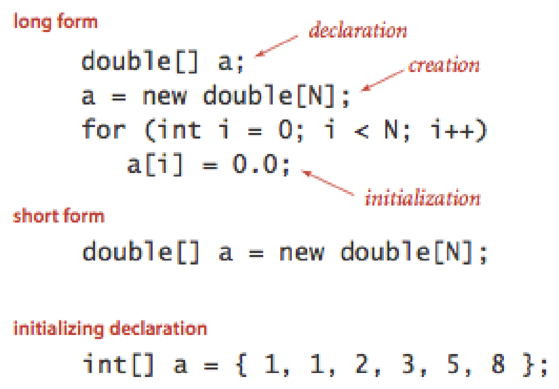
Most algorithms of interest involve organizing the data involved in the computation. Such organization leads to *data structures*, which also are central objects of study in computer science. A data structure organises the data in a certain way, in order to make sure that it is usable by a certain algorithm. Algorithms and data structures go hand in hand.

We will see this cooperation in this module. Like any craft (and yes, programming is a craft too), you can use tools and logical steps in order to produce something that works. Therefore, you can think of data structures as tools and algorithms as the necessary steps.

2 An Elementary Data Structure: Array

You have already learnt an elementary data structure either at work or when you attended basic programming modules: the array. An array stores a sequence of values that are all of the same type. We want not only to store values but also to access each individual value. The method that we use to refer to individual values in an array is numbering and then indexing them. If we have N values, we think of them as being numbered from 0 to N-1. Then, we can specify one of them in our code by using the notation a[i] to refer to the ith value for any value of i from 0 to N-1. This code construct is known as a *one-dimensional* array.

This is the notation of an array in Java:



```
long form
    double[] a;
    a = new double[N];
    for (int i = 0; i < N; i++)
       a[i] = 0.0;
short form
    double[] a = new double[N];

initializing declaration
    int[] a = { 1, 1, 2, 3, 5, 8 };
```

3 Creating And Using Arrays

Making an array in a Java program involves three distinct steps:

- Declare the array name and type.
- Create the array.
- Initialize the array values.

To declare the array, you need to specify a name and the type of data it will contain.
To create it, you need to specify its length (the number of values).
The keyword *new* in the second statement is a Java directive to create the array. The reason that we need to explicitly create arrays at run time is that the Java compiler cannot know how much space to reserve at compile time (unlike primitive - types).

Copying vs Aliasing

Remember, the name of an array refers to the WHOLE array. If we assign an array name to another array, both will refer to the same array:

```
int[] a = new int[10];
…
a[i] = 1234;
…
int[] b = a;
…
b[i] = 5678;   //a[i] is now 5678
```

This kind of assignment is called „Aliasing". Both refer (or point) to the same address.
Try it yourself:

Initiliaze an array as shown above and assign another array to the first. You can now compare the object-references like this:

```
System.out.println((Object)a.toString());

System.out.println((Object)b.toString());
```

Both will have the same object reference.

Copying

If you want to copy an array, you must declare, create and initialize a new array. And then copy all values from the original array to the new one.

Example:

```
int N = a.length;
double[] b = new double[N];
for (int i = 0; i < N; i++){
     b[i] = a[i];
}
```

Array Boundaries

The last element of an array a[] is always a[a.length-1]. Java does automatic bounds checking—if you have created an array of size N and use an index whose value is less than 0 or greater than N-1, your program will terminate with an *ArrayOutOfBoundsException* runtime exception.

Using Arrays

We will use arrays in the following exercises. Use *static* methods, since we are only interested in the functions in our program and not in using objects.

Example:

```
public static void printArrays(int[] numbers){

     for (int i = 0; i < numbers.length; i++){
          System.out.println(„current number is „ + numbers[i]);
     }
}
```

Obviously Java offers plenty of out-of-the-box library classes which cover some of the topics we are dealing with in this module (*Arrays*, *LinkedList*, sorting, etc.).
However, for the first few exercises we will implement these structures and algorithms ourselves, before using readymade classes.

User Interface: Text-based or Graphic-based

There are no restrictions how you should design your user interface. If you know Java Swing or Java Fx, you can implement small GUIs for user input and output. You can also just simply use the console. In that case, make sure to implement a separate *Input-Reader* class in order to make a distinction from the user interface to the actual algorithm and data structure.

Programming your own API

It is always good to consider that your own code might be re-usable. So see each program you write as a library implementation. Program with this approach:

Write code for the client, a top-level implementation that breaks the computation up into manageable parts.
■ Articulate an API for a library (or multiple APIs for multiple libraries) of static methods that can address each part.
■ Develop an implementation of the API, with a main() that tests the methods independent of the client.

Exercises

3.1 Exercise: Search for Minimum and Maximum

Write a program that can search for the minimum and maximum value in an array. Generate an array of length 10 (or 100) with randomly chosen values.

3.2 Exercise: Shifting Elements

Write a program that reads ten numbers from the user (with a separate user-input) and puts these into an array of fix lenght. We want to add each new element at the *beginning* of the array. Can we just work with one array?

3.3 Exercise: Returning Letters Reversed

Write a program that accepts a string and reverses each letter for the output:

```
Your input:  hello
My output: olleh
```

4 Elementary Sort-Algorithm: Bubble Sort

Sorting is one of the standard processes in IT and can be applied to small-scale data up to very large data volumes. There are many different sort-algorithms. We will look first at one of the classics, the *Bubble-Sort*, before considering other algorithms.

4.1 Doing the Bubble-Sort manually

Before we even consider programming this algorithm, we will play it through manually on paper. This gives you a good opportunity to track each processing step.

➔ See the separate worksheet.

4.2 Implementing Bubble-Sort

The steps: repeatedly step through the list to be sorted, compare each pair of adjacent items and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted.

In Pseudo-Code:

```
begin
    for i = N to 1 do
        for j = 2 to i do
            if array[j-1] > array[j] then
                e = array[j-1];
                array[j-1] = array[j];
                array[j] = e;
            end;
        end;
end;
```

Like „bubbles", the elements move slowly to the top (or bottom) of the list.

This comparison sort algorithm is not very practical with large data volumes, but it does its job and it illustrates how a few simple steps can be effective.

4.3 Write the code

Use the file „NumberInput.txt" and show the elements before and after the sort.

4.4 How many times does the sort run through?

You can easily answer this question by debugging the steps.

Your answer: ________________________________

4.5 Optimizing the Bubble Sort

The bubble sort algorithm can be easily optimized by observing that the *n*-th pass finds the *n*-th largest element and puts it into its final place. So, the inner loop can avoid looking at the last $n - 1$ items when running for the *n*-th time. (source: Wikipedia)

➔ Show your solutions to your teacher.

4.6 Sorting Larger Arrays and Measuring Performance

One crucial aspect when dealing with algorithms is their performance. A good way to measure this is by using the *currentTimeMillis*() method in the *System* class. Therefore you can calculate the elapsed time for the sort.

Initialise large arrays (10'000 or more) by adding these with random numbers, something like this:

```
Random random = new Random();
...
array[i] = random.nextInt(101);
```

Now measure the performance of the bubble sort. Compare your results with your colleagues.

5 Further Exercises with Bubble Sort

You can use the bubble sort for numerical and alphanumerical sorting, since the problem is for both the same (ie. 5 > 2 and D > B according to ASCII).

If we want to sort whole sentences alphabetically, we have to compare several characters with each other. Example: „Muheim‟ is before „Mutzer‟.

- ➔ Read in a file with 10 Strings and sort these alphabetically.
- ➔ Show the results as an output.

6 Advanced: Quicksort and Insertion Sort

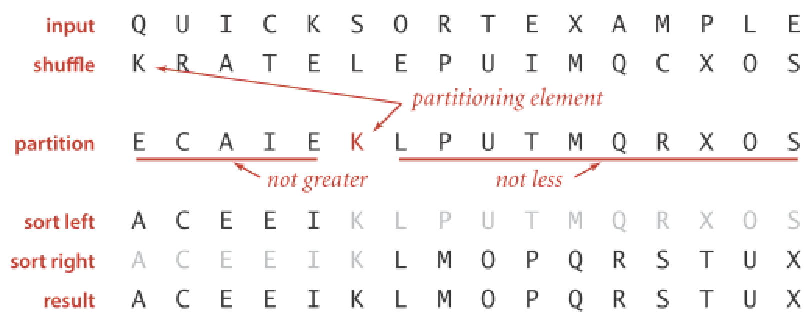
This section deals with two other sorting algorithms which are far more efficient than bubble sort.

6.1 Quicksort

The quicksort is a common sorting algorithm.
It is popular because it is not difficult to implement, works well for a variety of different kinds of input data, and is substantially faster than any other sorting method in typical applications.

The basic algorithm is a *divide-and-conquer method* for sorting. It works by *partitioning* an array into two subarrays, then sorting the subarrays independently.

See following image for the steps:


input Q U I C K S O R T E X A M P L E
shuffle K R A T E L E P U I M Q C X O S
partitioning element
partition E C A I E K L P U T M Q R X O S
not greater
not less
sort left A C E E I K L P U T M Q R X O S
sort right A C E E I K L M O P Q R S T U X
result A C E E I K L M O P Q R S T U X

Quicksort overview

In this example they implement a shuffle before partitioning the list.
Very important is picking the pivot. A wrong way would be to use the first element as the pivot. The better way is to choose the pivot randomly. After choosing the pivot, you have two partitions.

Task:

1. Work in pairs and do some research on how quick-sort can work.
2. Do a sketch of the logical process (in form of a diagram, activity diagram or structogram).
3. Implement your version. Compare it to the bubble sort by useing the same data.

6.2 Insertion Sort

Another sorting algorithm is the insertion sort. This works similar to bubble sort, since both are „in place" sorting algorithms.

However, there is one big difference:

Insertion sort is like sorting a hand of cards. You first sort first 2 cards. Then place the 3rd card in its appropriate position in the 2 sorted cards. Then 4th card is added to sorted 3 cards in the correct position and so on.

With bubble sort you are dealing with the whole list and comparing pairs in each step.

Group Work:

Do some research on how insertion sort works and show an implementation.