Algorithms

Learning Objectives

- You understand the concept of algorithms and you are familiar with a few algorithms
- You can explain how selection sort works
- You can explain how the linear search and binary search algorithms work

Algorithms, precise instructions on how to to accomplish a specific task, are at the core of computer science. In the context of programming, algorithms are typically defined using source code.

The concept of *efficiency* is often associated with algorithms. A programs efficiency, i.e, the computation of required information fast enough, is an integral part of a programs usability. If it took two days for an algorithm designed for forecasting tomorrows weather run, the results wouldn't be very useful! Similarly, a user viewing a TVs program guide won't get any use out of it, if the tv-shows info only loads after the show already ended.

In a more general sense, retrieving and displaying information quickly is an integral part of any applications function. Next let's explore algorithms associated with retrieving and sorting information. While the following examples utilize arrays, the algorithms shown will also work with other data-structures meant for storing information, such as lists.

Sorting information

If the information (data) giving to a computer, doesn't follow any rules and isn't in order, retrieving that information is taxing for the computer. We need order!

Selection sort

Every programmer should be familiar with at least one sorting algorithm (i.e a way to sort an array). Let's familiarize ourselves with one "classic" sorting algorithm, the selection sort. We'll do so with a programing exercise.

Programming exercise:

Sorting (5 parts)

Points
5/5

NB! By submitting a solution to a part of an exercise which has multiple parts, you can get part of the exercise points. You can submit a part by using the 'submit' button on NetBeans. More on the programming exercise submission instructions: Exercise submission instructions.

Part 1: Finding the smallest value

Create in the class `MainProgram` a class method `smallest` that takes an integer array as a parameter. It should return the smallest value in the array.

Here is the structure of the method:

```
public static int smallest(int[] array){
    // write your code here
}
```

The next example illustrates how the method works:

```
int[] numbers = {6, 5, 8, 7, 11};
System.out.println("Smallest: " + MainProgram.smallest(numbers));
```

Sample output

```
Smallest: 5
```

Part 2: Index of the smallest value

Create a method called `indexOfSmallest` in the class MainProgram. It should return the index of the smallest number in the array that it receives as a parameter.

Here is the structure of the method:

```
public static int indexOfSmallest(int[] array){
    // write your code here
}
```

The following code illustrates how to use the method:

```
// indices:      0  1  2  3  4
int[] numbers = {6, 5, 8, 7, 11};
```

```
System.out.println("Index of the smallest number: " + MainProgram.indexOfSmallest(numbers
```

Sample output

```
Index of the smallest number: 1
```

The smallest number in the array is 5, and its position in the array (i.e. index) is 1. Be sure to remember that indexing an array begins at 0.

Part 3: Index of the smallest value after a certain value

Create in the class MainProgram a class method called `indexOfSmallestFrom`. It works similarly to the method in the previous section, but only considers the table values from a certain index forwards. In addition to the table, it receives this start index as a parameter.

The structure of the method is the following:

```
public static int indexOfSmallestFrom(int[] table, int startIndex) {
    // write your code here
}
```

The following code illustrates how the method words:

```
// indices:       0  1  2  3   4
int[] numbers = {-1, 6, 9, 8, 12};
System.out.println(MainProgram.indexOfSmallestFrom(numbers, 0));
System.out.println(MainProgram.indexOfSmallestFrom(numbers, 1));
System.out.println(MainProgram.indexOfSmallestFrom(numbers, 2));
```

Sample output

```
0
1
3
```

In this example the first method call searches for the index of the smallest number, starting from index 0. Starting from index 0, the smallest number is -1 and its index is 0. The second method call searches for the index of the smallest value starting from index 1. In this case the smallest number is 6 and its index is 1. The third calls searches for the index of the smallest value starting at index 2. Then the smallest number is 8 and its index is 3.

Part 4: Swapping numbers

Create a class method `swap` in the class MainProgram. It receives as its parameters an array and two indices inside it. The method swaps the numbers in these indices with each other.

The basic structure of the method is:

```
public static void swap(int[] array, int index1, int index2) {
    // write your code here
}
```

The following illustrates how to use the method. To print an array we take use of the `toString` class method of the class `Arrays`. It formats an array into an easily readable string.

```
int[] numbers = {3, 2, 5, 4, 8};

System.out.println(Arrays.toString(numbers));

MainProgram.swap(numbers, 1, 0);
System.out.println(Arrays.toString(numbers));

MainProgram.swap(numbers, 0, 3);
System.out.println(Arrays.toString(numbers));
```

Sample output

```
[3, 2, 5, 4, 8]
[2, 3, 5, 4, 8]
[4, 3, 5, 2, 8]
```

Part 5: Sorting

We have now assembled a set of useful methods. With their help, we can implement a sorting algorithm known by the name of selection sort.

The idea of selection sort is:

- Move the smallest number in the array to index 0.
- Move the second smallest number to index 1.
- Move the third smalles number in the array to index 2.
- Etc.

In other words:

- Examine the array starting from index 0. Swap the following two numbers with each other: the number at index 0, and the smallest number in the array starting from index 0.

- Examine the array starting from index 1. Swap the following two numbers with each other: the number at index 1, and the smallest number in the array starting from index 1.
- Examine the array starting from index 2. Swap the following two numbers with each other: the number at index 2, and the smallest number in the array starting from index 2.
- Etc.

Implement a class method called `sort` based on the idea above in the class MainProgram. It should include a loop that goes through the indices of the array. Certainly the method `indexOfSmallestFrom` and `swap` will come in handy. Additionally, print the contents of the array before sorting and after every iteration of the loop to ensure that the algorithm works as you expect it to.

The definition of the method looks like this:

```
public static void sort(int[] array) {

}
```

Use at least the following example to test how the method functions:

```
int[] numbers = {8, 3, 7, 9, 1, 2, 4};
MainProgram.sort(numbers);
```

The output of the program should look like the print below. Observe that you must print the contents of the array after each swap!

Sample output

```
[8, 3, 7, 9, 1, 2, 4]
[1, 3, 7, 9, 8, 2, 4]
[1, 2, 7, 9, 8, 3, 4]
[1, 2, 3, 9, 8, 7, 4]
[1, 2, 3, 4, 8, 7, 9]
[1, 2, 3, 4, 7, 8, 9]
[1, 2, 3, 4, 7, 8, 9]
```

Mark how the array becomes sorted little by little starting from the beginning and advancing towards the end of the array.

Exercise submission instructions

How to see the solution

Static or not

At the start of the course, all of our methods included the `static` modifier, but when we started using objects, use of the `static` modifier was banned.

Methods in Java can be divided into two groups, based on whether they have the `static` modifier or not. Methods without the `static` modifier are **instance methods**. Methods with the `static` modifier are **class methods**

Instance methods are methods that are associated with an object, can process the objects variables and can call the object's other methods. Instance methods specifically CAN use the `this` modifier, which refers to the variables associated with the specific object, that is calling the instance method. Class methods can't use the `this` modifier, meaning that they can only access the variables they are given as parameters or that they create themselves.

In reality class methods can also access class variable, among other things. However, these things are outside the scope of this course.

Built-in sorting algorithms in Java

Java offers a significant amount of ready to use sorting algorithms. Arrays can be sorted (into their natural order) using the class method `sort` of the `Arrays`-class. Lists can be sorted (into their natural order) using the class method sort of the `Collections` class.

```
int[] numbers = {8, 3, 7, 9, 1, 2, 4};
System.out.println(Arrays.toString(numbers));
Arrays.sort(numbers);
System.out.println(Arrays.toString(numbers));
```

Sample output

```
[8, 3, 7, 9, 1, 2, 4]
[1, 2, 3, 4, 7, 8, 9]
```

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(8);
numbers.add(3);
numbers.add(7);
System.out.println(numbers);
Collections.sort(numbers);
System.out.println(numbers);
```

Sample output

```
[8, 3, 7]
```

```
[3, 7, 8]
```

Java's built-in sorting algorithms work with value type variables and some of Java's built-in reference type variables, like String. In order for our own classes to be sorted, we need to provide Java with some tips on how to do that, because the classes themselves don't contain information on how objects created from them should be ordered. We'll get back to ordering objects created from classes we made ourselves in the advanced course in programming.

Programming exercise:
Ready-made Sorting Algorithms

Points
1/1

Add the following methods to the Main class:

- `public static void sort(int[] array)` that sorts an array of integers.
- `public static void sort(String[] array)` that sorts an array of strings.
- `public static void sortIntegers(ArrayList<Integer> integers)` that sorts a list of integers.
- `public static void sortStrings(ArrayList<String> strings)` that sorts a list of strings.

Be sure to use the ready-made Java libraries in your implementation.

Exercise submission instructions

How to see the solution

Information retrieval

Next let's take a look at algorithms meant for information retrieval.

Linear search

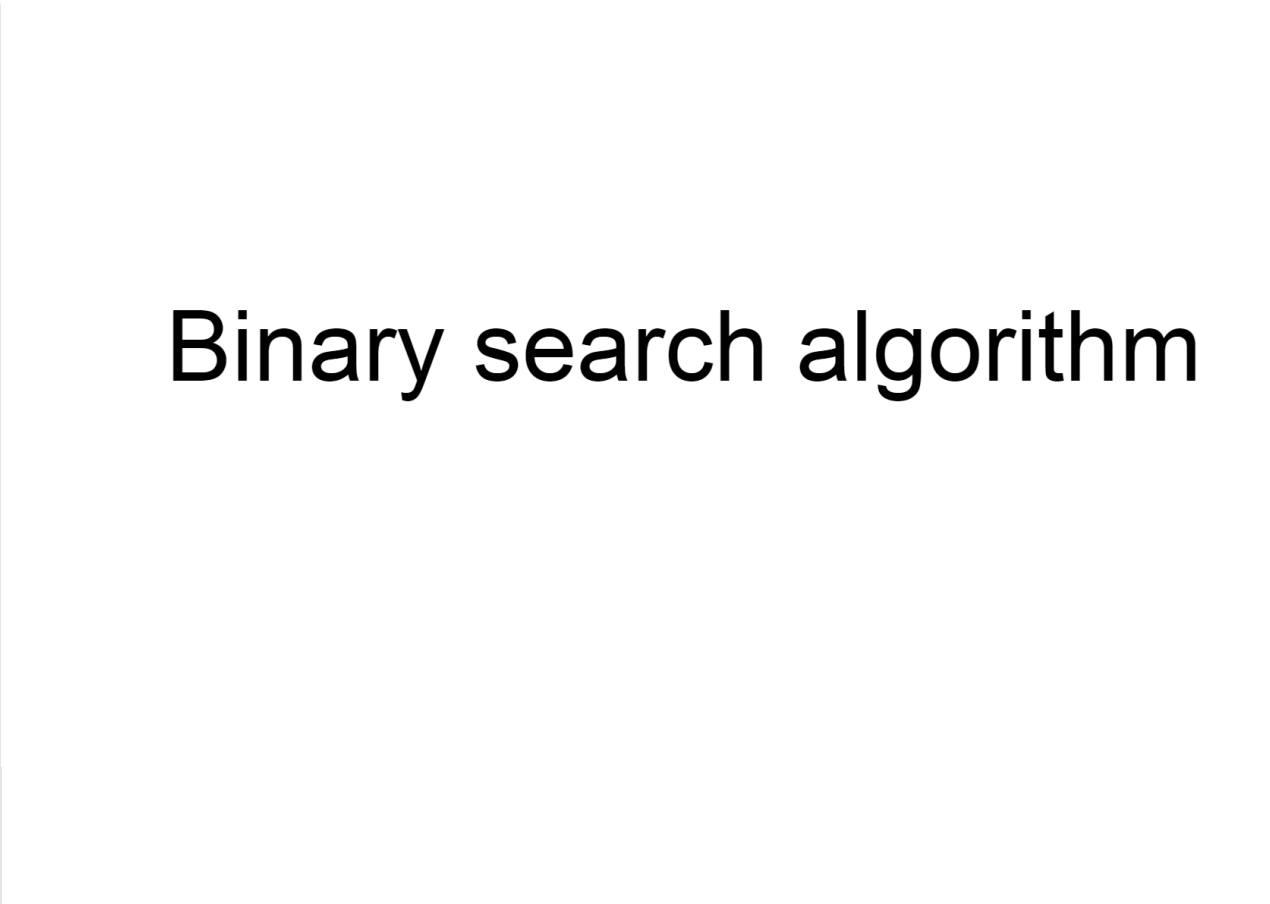
Linear search is a search algorithm that searches for information in an array by going through every value in the array one by one. When the value that was searched for is found, its index is immediately returned. If the requested value cannot be found, linear search returns the information that the value was not found — typically this means returning `-1` instead of a valid index.

```
public class Algorithms {

    public static int linearSearch(int[] array, int searched) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == searched) {
                return i;
            }
        }

        return -1;
    }
}
```

In the worst case scenario, i.e when the value searched for isn't found, the algorithm has to do as many comparisons as there are values in the array. In an array containing, say, 10 million values, this means 10 million comparisons. If we are doing more than one search, it makes sense to try and improve efficiency.

Binary search (aka half-interval search or logarithmic search)

When the data searched is in order, searching can be implemented a lot more efficiently than in linear search. The idea behind Binary Search is to start looking for the searched value in the middle index of the array (or list), compare the value found there to the searched value, and if needed (i.e, when the value isn't found there) eliminate half of the search area. The algorithm is more thoroughly introduced in the following slideshow.



Programming exercise:
Searching

Points
2/2

NB! By submitting a solution to a part of an exercise which has multiple parts, you can get part of the exercise points. You can submit a part by using the 'submit' button on NetBeans. More on the programming exercise submission instructions: Exercise submission instructions.

In the exercise template you'll find the class `Book`, ready for use. The class describes objects with a numeric id `id` and a name `name`.

In this exercise you will implement linear search and binary search algorithms for searching for books by their numeric id. In the exercise template you'll find the names of the methods to be implemented ready for you — at the moment these method always return -1 — you'll also find the Main method ready to be used for testing your methods.

Part 1: Linear search

The linear search algorithm works by checking every value in a list or an array one at a time, starting from index 0.

In the `Main`-class, implement a method `public static int linearSearch(ArrayList<Book> books, int searchedId)`, which searches the list it received as a parameter, for a book with an `id` variable that matches the value of `searchedId` variable it received as a parameter. If that book is found, the method, should return the index it's located at in the list it received as a parameter. If the book isn't found, the method should return the value `-1`.

Part 2: Binary search

In the `Main`-class, implement a method `public static int binarySearch(ArrayList<Book> books, int searchedId)`, which searches the list it received as a parameter, for a book with an `id` variable that matches the value of `searchedId` variable it received as a parameter. If that book is found the method, should return the index it's located at, in the list it received as a parameter. If the book isn't found, the method should return the value `-1`.

The method must be implemented as a binary search, which assumes the list is ordered. You should also assume, that the `id`s towards the beginning of the list,

are always smaller than the `id`s towards the end of the list.

To help you, you have the idea of the binary search shown in the slideshow above, as well as the *pseudocode* for binary search. Pseudocode is a description of a programs/methods function written in a programming-language like way.

In the slideshow above, the idea of a binary search was described as follows:

- The array or list searched must be sorted
- The search begins in the middle of the array or list
- If the value in the middle-point being examined isn't the searched value, eliminate half of the searched area, and move on to examine the middle-point of the remaining half.
- If the value in the middle-point being examined is the searched value, return the index of the middle-point being examined.
- If there is nowhere left to search (the entire area has been eliminated), return the value -1, indicating that the searched value was not found (i.e, it wasn't in the list or array searched).

The pseudocode for binary search looks like this:

```
// assuming the variable searched exits
// assuming the variable list exits
begin = 0 // the 0th index of the list (i.e, the first index of the list)
end = size(list) - 1 // the last index in the list

repeat until begin is larger than end:
    middle = (end + begin) / 2

    if the value at list[middle] is searched
        return the value of the variable middle

    if the value at list[middle] is smaller than searched
        begin = middle + 1

    if the value at list[middle] is larger than searched
        end = middle - 1

return value -1
```

Note that in the case of books, you are examining the values of the books' `id`-variable. Meaning that in this exercise, instead of examining the value at an index, you should examine the value of the `id`-variable of the value found at the index.

Exercise submission instructions

How to see the solution

You have reached the end of this section! Continue to the next section:

→ 3. Larger programming exercises

Remember to check your points from the ball on the bottom-right corner of the material!

In this part:

Source code of the material

This course is created by the Agile Education Research -research group of the University of Helsinki.

Credits and about the material.



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

MASSIIVISET AVOIMET VERKKOKURSSIT
MASSIVE OPEN ONLINE COURSES · MOOC.FI