Lab5 - Sorting Algorithms

Important! Read the document "Information and rules about the labs". Read through the entire lab specs before starting to write code so that you get an idea of what to do, the extent and can plan your work and time. Read each assignment carefully: it is important that you follow the instructions to get everything done correctly. If something is unclear, contact your lab assistant: Gabriele Capannini (`gabriele.capannini@mdh.se`) or Abu Naser Masud (`masud.abunaser@mdh.se`). You need to complete all the lab assignments to be eligible to attend the final lab examination.

Task 1: Sorting

In this laboratory you will write and evaluate at least three sorting algorithms.

Task 1.1: Download the project

Download and extract the `Lab5.zip` file in this directory. Create a new project with a proper name in your development environment and add all files in the zip file to the project. Make sure you can compile the project without any error messages (there may be warnings: just ignore them at the moment).

This lab skeleton is actually more than one skeleton: it's a whole program that sorts and prints results of a variety of sortings. You do not need to understand how it works if you do not want to, but it is recommended. You need to implement a small number of functions described in the following subtasks.

Task 1.2: Determine which sorting algorithms you want to implement

We have talked about the following five sorting algorithms:

- Bubblesort
- Insertionsort
- Selectionsort
- Mergesort
- Quicksort

You must implement at least three of the above algorithms (it is of course allowed to implement four or all five as well). However, one of them must be Mergesort or Quicksort.

Task 1.3: Implement search algorithms

There are empty function bodies for all the sorting algorithms in the *SortingAlgorithms.c* file. They all take the following three arguments:

- *arrayToSort*: the array to be sorted. This is an array of **unsigned int** (but **typedef** to *ElementType*).
- *size*: the size of the array. (type **size_t** is a standard type and is actually an **unsigned int**)
- *statistics*: This is a pointer of a type that will keep track of statistics for the number of bytes used and the number of comparisons performed by your algorithms. You do not need to know what the type looks like to use it.

Implement your sorting algorithms. Of course, it is allowed to add **static** auxiliary functions. For the more advanced algorithms like *mergesort* and *quicksort*, it will be necessary to do this because they consist of several steps and are recursive.

Statistics In order for the statistics to work correctly, you need to use some functions contained in this system. In your algorithms, you should never use comparison operators (e.g. $<, >, ==, !=$), but use the functions *lessThan()*, *greaterThan()*, *equalTo()*, etc. instead, which are declared in *statistics.h*. These functions return 1 (true) if the comparison matches and 0 (false) otherwise. They also take a pointer to the statistics variable as an argument and update it. Note that you will also use these in loops. So a loop that runs 10 times will look like:

```
for (i = 0; lessThan(i, 10, statistics); i ++)
```

Similarly, you should use the *swapElements()* function (also defined in *statistics.h*) to swap two elements. If you have done correctly, the program will keep track of how many comparisons and changes your algorithms are doing.

In order for the program to work: When you want to test that your sorting function works properly, you must tell the program that the sorting function exists. You do this by describing the correct line in the *isImplemented()* function at the top of the *sortingAlgorithms.c*. You can now run the program, if your function does not sort correctly, the program will tell you (you will receive a message and an **assert()** will be activated).

When you run the program, statistics for your sorting algorithms will end up in a text file. Each algorithm will sort three different sizes and three different sorts (unsorted, sorted, and rearranged). Statistics for the number of bytes and the number of comparisons for these will be written to *result.txt* that will be in your project folder. If you want to enter the entire path to where the file is to be saved, then change *line 18* in the *main.c* file.

Open *result.txt* and familiarize yourself with the result. Try to relate the numbers to the algorithms. Does it seem to be what we said in the lecture? Which algorithms work well for sorting the arrays?