

Opinnäytetyön nimi  
Tarvittaessa alaotsikko

Vilho Mäkeläinen

Sorting algorithm project

April 2024

TIIVISTELMÄ

Tampereen ammattikorkeakoulu

Tietotekniikka (Insinööri)

Ohjelmistokehitys

Vilho Mäkeläinen:

Lajitteluprojekti

Huhtikuu 2024

Projektissa toteutetaan ja demonstroidaan haku ja lajittelualgoritmejä

ABSTRACT

Tampereen ammattikorkeakoulu

Tampere University of Applied Sciences

Banchelors of Applied Sciences

Software engineering

Vilho Mäkeläinen:

Sorting algorithm project

April 2024

A project to demonstrate and implement different search and sorting algorithms.

INDEX

[TARGET GRADE 5](#_Toc165342526)

[1 STRUCTURE 6](#_Toc165342527)

[1.1 Return structure 6](#_Toc165342528)

[1.1.1 Github structure 6](#_Toc165342529)

[2 RUNNING 7](#_Toc165342530)

[2.1 With g++ 7](#_Toc165342531)

[2.1.1 With make 7](#_Toc165342532)

[3 GRADES 8](#_Toc165342533)

[3.1 Grade 1: 8](#_Toc165342534)

[3.1.1 Tests 9](#_Toc165342535)

[3.2 Grade 2 10](#_Toc165342536)

[3.2.1 Tests 14](#_Toc165342537)

[3.3 Grade 3 16](#_Toc165342538)

[3.3.1 Tests 18](#_Toc165342539)

[3.4 Grade 4 19](#_Toc165342540)

[3.4.1 Tests 21](#_Toc165342541)

[3.5 Grade 5 22](#_Toc165342542)

[3.5.1 Tests 26](#_Toc165342543)

[4 WORK 29](#_Toc165342544)

[4.1 Day 1 29/04/2024 29](#_Toc165342545)

[4.2 Day 2 30/04/2024 29](#_Toc165342546)

# TARGET GRADE

I’m targeting grade 5 with this project.

# STRUCTURE

## Return structure

Following the instructions given I have put everything in one directory so you can build the thing by the requirements.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto

Kuvaus luotu automaattisesti

### Github structure

This repo should have another branch with more proper project structure and makefile to do it correctly.

# RUNNING

## With g++

Use the bash command “g++ \*.cpp -o testped” in this project folder to compile the program as requested. To run it use: “./testped” in a Linux terminal to run it.

### With make

Navigate to the correct folder with your terminal and use the following command:

“make” to build the project, then run “./testped” or “make run” to run the project.

# GRADES

## Grade 1:

This milestone was simple to implement. Originally, I just did a simple C++ program with normal main methods and functions. The sequential search algorithm is very simple to implement using normal arrays and indices. We just iterate over the array and compare to the previous values to find the desired value. The data generation was simple to implement as well, just a loop and an if statement were needed to do it.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto

Kuvaus luotu automaattisesti

The sequential search algorithm is in its own method:

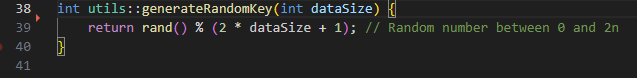
Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, Multimediaohjelmisto

Kuvaus luotu automaattisesti

I use an external utils class for the random data generation:

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti



### Tests

With random index and 10 indices:

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti, ohjelmisto

Kuvaus luotu automaattisesti

With 10 elements and an invalid custom element

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus, typografia

Kuvaus luotu automaattisesti

## Grade 2

This was quite straightforward to implement as well. I just needed to create the binary search algorithm and make comparisons between the two.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto

Kuvaus luotu automaattisesti

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto

Kuvaus luotu automaattisesti

I love the idea behind the binary search algorithm, it’s a very logical way to find something quickly. Just start from the middle, test if the value is lower than current use that half of the list and split it again. If you do it enough the wanted value will be found fast. The only issue is that the list needs be sorted beforehand. In this case we are using the same odd number list than before.

Here is the binary search class:

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, Käyttöjärjestelmä

Kuvaus luotu automaattisesti

We are now using the time classes as well to determine the time difference between the two:

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, näyttö

Kuvaus luotu automaattisesti

### Tests

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto

Kuvaus luotu automaattisestiKuva, joka sisältää kohteen teksti, kuvakaappaus, ruokalista, Fontti

Kuvaus luotu automaattisesti

Here we ran the test with array size 1000000 and 10 repetitions to get more accurate average results.

We can see that especially with very large numbers the binary search destroys the sequential search in time and steps. It’s so fast that the program registers it as 0.

## Grade 3

The insertion sort requires way more math and logic to implement compared to the search algorithms described above. At this point I started to restructure the whole project to be more in line of what it is now in the screenshots, moving things to their own classes, adding helper utils etc. This took some time to do but was well worth it. This was a good point to do it since the project changed to sorting algorithms from search algorithms.

Test function:

Kuva, joka sisältää kohteen teksti, kuvakaappaus

Kuvaus luotu automaattisesti

We are using a random data generator from the utils now:

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, Fontti

Kuvaus luotu automaattisesti

And a method to print a portion of the data to screen:

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti, ohjelmisto

Kuvaus luotu automaattisesti

Here is the insertion sort method itself:

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto

Kuvaus luotu automaattisesti

The comments above describe the process well. We just start from the first element and compare backwards. If the value is less than previous, we just move it backwards until it is not.

### Tests

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

Here we choose an array size 1000, so a pretty big array and show 20 elements before and after the sorting. We can see that the number get sorted wit a lot of comparisons and assignments. The issue with insertion sort is that it requires comparing to a lot of other elements to complete.

## Grade 4

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto

Kuvaus luotu automaattisesti

We do the same things as previously, but this time with quicksort. Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto

Kuvaus luotu automaattisesti

We use recursive method calls to perfom the search.

Quicksort is way less easy to understand at first. The basic idea is this:  
You get a pivot point eg. The last element of the array and split the array based on values that are lower than the pivot and higher than the pivot. We keep the pivot at the center. We then perform the same operation on the smaller sized number and higher sized numbers till we run out of them. At this point the numbers are sorted. It’s close to a binary search in the sense that we split the array and look at the halves of the data.

Here is a good image explaining the flow from favtutor: https://favtutor.com/blogs/quick-sort-cpp

Kuva, joka sisältää kohteen diagrammi, kuvakaappaus, viiva, teksti

Kuvaus luotu automaattisesti

### Tests

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

Here we can see that the run time of the quick sort is about ten times as fast as the insertion sort above. The worst-case time for quicksort can be bad n^2 but the average performance is very good n\*log(n)

## Grade 5

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, näyttö

Kuvaus luotu automaattisestiKuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto

Kuvaus luotu automaattisesti

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus, viiva

Kuvaus luotu automaattisesti

Here we are comparing the time between slower n^2 algorithm with a faster n \* log (n) algorithm.

We are using insertion sort and new merge sort for the example.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, Multimediaohjelmisto

Kuvaus luotu automaattisestiKuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, näyttö

Kuvaus luotu automaattisestiKuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

Merge sort works by continuously dividing the array of numbers in two until the elements can’t be divided any more. It the compares the two adjacent values flips them if the other is smaller than the other and merges them together. It does it to all pairs and then does the same for the rest of the groups. The groups keep getting larger and larger until they are all sorted.

Using ofstream I wrote the data to two csv files, and using the pandas library for python I plotted the results against each other. Here we can clearly see how well the merge sort algorithm works. The insertion sort is getting exponentially longer as the number of elements increases.

Kuva, joka sisältää kohteen teksti, Tontti, viiva, diagrammi

Kuvaus luotu automaattisesti

Here are screenshots from the data spreadsheets:

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisestiKuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, Fontti

Kuvaus luotu automaattisesti

And here is the python code I used to visualize the data with:

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto

Kuvaus luotu automaattisesti

### Tests

Below is a output of the results:

Kuva, joka sisältää kohteen teksti, ruokalista, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

The list is very long since there are so many samples, here is the bottom end of the ouput:  
Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti, ruokalista

Kuvaus luotu automaattisesti

We can observer that the time is increasing more and more.

# WORK

## Day 1 29/04/2024

I did about 7 hours of work and completed grades 1-3.

## Day 2 30/04/2024

I did about 5 hours of work and completed grades 4-5.