

DM548/DM588

Computer Architecture (& System Programming)

Fall 2024

Assignment 1: Sorting in Assembly

In this assignment we will emulate a small research project on efficient implementation of a program to solve a computational problem. There are therefore 3 parts:

1. Programming, where you after deciding on an algorithm will implement it along with the supporting code for input and output.
2. Implementation evaluation, where you will perform systematic benchmarks of your implementation. You will also need to make small code modifications to obtain statistics of your program performance.
3. Report, where you will document the whole project. Importantly this should be written as if this project description does not exist and was all your own idea.

1 Programming

Your task is to write a program that when given a list of (x, y) -coordinates it sorts them on just the y -dimension and then prints the result. The program must be implemented in x86-64 Linux assembly. This means that you may not link to any libraries or use a higher-level language to generate the assembly code. You may use code you have written for the lab exercises, as well as the code provided for the labs. Additionally, some code snippets especially for this project are provided, which you may use and modify as you much as you like. You are free to select which sorting algorithm(s) you implement, but you are recommended to start with an a relatively simple in-place algorithm (e.g., insertion sort) to quickly obtain a working solution. Afterwards you can experiment with better algorithms if time permits.

1.1 Input/Output Specification

Your program should be callable with a single command-line argument which is a filename for a file with coordinates to sort.¹ The indicated file will contain 1 coordinate per line with the format:

$\langle x \rangle \backslash \mathbf{t} \langle y \rangle \backslash \mathbf{n}$

where

- $\backslash \mathbf{t}$ is a tab character (ASCII code 9),
- $\backslash \mathbf{n}$ is a newline character (ASCII code 10), and
- $\langle x \rangle$ and $\langle y \rangle$ are integers in the range 0 to 32767, i.e., the non-negative numbers representable in 16-bit twos complement format.

Note that the line endings are the customary ones on Linux, but that Windows and macOS usually uses different characters for line endings.

Your program must read the numbers in the file, sort them on the y -coordinate (i.e., the second number), and then print them to stdout in the same format as the input file. Nothing else may be written to stdout, but you are free (and encouraged) to print error messages to stderr. This means that

```
> ./yourProgram numbers.txt
```

should do the same as

```
> sort -n -k 2 numbers.txt
```

(Though, if two coordinates have the same y -value, their order may be different than what **sort** produces.)

1.2 Hints

Iteratively Obtaining a Working Solution

It may be tempting to write the program in the order it should work in the end, but it is difficult to test it as you go along. Instead, consider breaking the program into smaller bits and work “outside-in”. For example, if you implement something that reads input, implement output functionality for it to check that it works. The last major part should be the sorting algorithm it self.

Creating Random Numbers

For creating a test file with, say 42, coordinates you can use the **\$RANDOM** variable in Bash:

¹You are welcome to implement parsing of extra command line switches but they must be optional.

```
> for i in {1..42}; do echo -e "$RANDOM\t$RANDOM"; done > numbers.txt
```

1.2.1 Checking the Output

You can use the `check.py` Python program available on itslearning to check if your program output is sorted:

```
> ./yourProgram numbers.txt > output.txt  
> python3 check.py numbers.txt output.txt
```

It will check 1) whether the second file (“`output.txt`“ in the example above) is sorted, and 2) whether the lines in the two files are permutations of each other. If the first check, for sorted order, fails it will print the line numbers of the two first lines that are not in order, along with the lines, e.g.:

```
> python3 check.py numbers.txt output.txt  
Disorder between line 42 and 43. Lines are  
26263 31002  
15812 11169
```

If the second check, for permutation, fails it will print the first lines it can not match up.

The second check can also be performed manually with ordinary command line tools:

```
> ./yourProgram numbers.txt > output.txt  
> sort numbers.txt > sorted_numbers.txt  
> sort output.txt > sorted_output.txt  
> diff -u sorted_numbers sorted_output.txt
```

1.2.2 Code Snippets

Through itslearning you can download snippets with the following functions. They all follow the ordinary calling conventions for x86-64 on Linux. You are additionally allowed and encouraged to use snippets from the lab exercises and your solution code.

allocate (`allocate.s`) This function asks the OS to change the size of the data segment in the running program, thereby getting more space. As argument you simply pass the number of bytes needed in `%rdi`. A pointer to the allocated memory is returned in `%rax`.

Note that memory is never freed, so there really is not much management of memory, and you should be careful if you allocate memory in a recursive function.

getFileSize (`fileHandling.s`) When you have opened a file you can get its size (in bytes) with this function. Pass the file descriptor for the file as argument.

getLineCount (`parsing.s`) When you have read the contents of a file into a buffer, this function can count how many coordinates are in it (by counting the number of newline characters). This will allow you to allocate the appropriate amount of memory for the parsed coordinates.

- The first argument must be the address of the memory with the data.
- The second argument must be the length of that piece of memory.

parseData (`parsing.s`) This function converts the ASCII representation of the coordinates into pairs of numbers.

- The first argument must be the address of the text buffer to parse.
- The second argument must be the length of that text buffer.
- The third argument must be a pointer to a piece of memory to write the parsed coordinates to.

Let n be the number of coordinates in the input data, and let (x_i, y_i) denote the i th coordinate. The parsed coordinates will then be written into the the given memory block as an array with the content $x_0, y_0, x_1, y_1, x_2, y_2, \dots, x_{n-1}, y_{n-1}$. Each individual number will be taking up 8 bytes (make sure you have allocated enough memory).

Part 2: Program Evaluation

After you have finished your sorting program, you should evaluate its performance, e.g., to compare it with the theoretical complexity of your chosen algorithm(s). The goal is to perform benchmarks to convince the reader of your report that your implementation performs as expected.

As a default suggestion for this assignment:

1. Generate test instances with each of the following amount of uniformly random coordinates:
 - 10,000
 - 50,000
 - 100,000
 - 500,000
 - 1,000,000
 - 5,000,000 (if the runtime is not unreasonably high)

For each size you should create 3 instances.

2. Measure the runtime of your program using the “`time`” command. You should record the “real”/“wall” time. If it differs dramatically from the “user” time, it would proper to discuss it in your report. Note, the displaying of output from the program may slow it down. To be sure this is not the case, you can redirect the output to the null device:

```
> time ./yourProgram numbers.txt > /dev/null
```

3. Display your runtime evaluation in an appropriate manner suitable for evaluating the performance.
4. Obtain statistics to compare the performance with the theoretical complexity of your sorting algorithm: Count the number of compare operations you use during sorting, where elements from the input participate.² How is the number of comparisons related to the overall runtime? Hints: In order to calculate the number of comparisons, you can simply increment a variable after each compare instruction and output the result in the end. Make sure you make your time measurements without this counting overhead. The submitted version of your program sources should not contain this counting or additional output either.
If your chosen sorting algorithm is not based on comparisons, you may skip the part about counting compare instructions (and the related tasks). You must instead illustrate how the execution time relates to the theoretical complexity of your chosen algorithm in an appropriate manner.
5. Related the number of compare instructions to the execution time. Calculate the number of comparisons per second as a performance metric for each of your input files. How does it change with different input sizes?
6. Assuming the sorting took no time (e.g., temporarily comment the call to your sorting algorithm out), the remaining runtime is the overhead of the program. How is the overhead time related to the file size?

Part 3: Report

As default you should write your report in `LATEX`, as is the standard in most of Computer Science and Mathematics, due to, e.g., its good support for typesetting mathematics and code. The following (non-exhaustive) list of packages may be of relevance:

²That is, comparisons for loops should not be counted.

- `hyperref`, for making references clickable.
- `pgfplots`, for rendering plots of data.
- `tikz`, for creating illustrations.
- `listings` or `minted`, for typesetting source code.
- `booktabs`, for creating prettier and more readable tables.
- `siunitx`, for typesetting numerical data and units.
- `algorithm2e`, for typesetting pseudo code.

Your report must be at most 8 pages long, including potential appendices but excluding potential references and front page. In general, the target audience of the report should be assumed to be someone who just went through an imaginary version of this course where the this project does not exist. That is, the intended reader knows basic algorithms and data structures as well as the complete DM548/DM588 course curriculum, including x86-64 Linux assembly, but has not read this project description. The report should in a clear and concise manner

- introduce the problem being studied,
- describe interesting design decisions, including your motivation for choosing the specific sorting algorithm you have implemented,
- describe interesting implementation details, with an appropriate amount of code snippets,
- present the results of your program evaluation in a structured, scientific way, equivalent to following the instructions above, and
- comment and discuss the results (not only present them).

You may write your report in either Danish or English, whichever you prefer to allow for the best result.

Practicalities

Working in Groups or Alone

You are allowed, and encouraged, to work in groups up to size 3. It is not recommended that you work alone.

When working in a group you are collectively responsible for the hand-in. For the report that in particular means you should work together on the whole document to obtain the best result.

Submission Format

Submission will be handled through itslearning. You create your groups yourself, as you desire. Even if you hand-in alone, you have to create a group with you as the only member. See the screenshot of process at the end of the document. Since itslearning is also changing, please let us know in case there are troubles with creating groups and we will find a solution.

Please upload a zip-File with the content and structure explained below:

Report

Your zip-File must have the following content.

Either:

- **report/**
The folder where your report sources resides. Generally all files used to compile the report should be in this folder somewhere, though data tables may be included from other folders.
- **report/report.tex**
The main L^AT_EX file for your report. It must be possible to compile it using “**pdflatex -shell-escape report.tex**” in the **report/** folder.³ Please do not commit the auto-generated files from L^AT_EX, not even the compiled PDF (the lecturer will compile it from your sources after the deadline).
- **report/*.bib** (optional)
If you use BibTeX for formatting your references, the database file(s) should be in the root of the **report/** folder. Thus, the report can then be fully compiled with something like

```
pdflatex -shell-escape report.tex \  
&& bibtex report.aux \  
&& pdflatex -shell-escape report.tex \  
&& pdflatex -shell-escape report.tex
```

or, in case you really do not want to use L^AT_EX:

- **report.pdf** (note: not in a subfolder)
Your report in PDF format.

If **report.pdf** is present, it will be assumed to be the report to be evaluated, otherwise the one that can be compiled from the **report/** folder will be evaluated.

³The use of the **-shell-escape** flag is generally not needed, but it used on the test server to allow use of certain packages.

Program Sources

Pack your source code into your zip-file as follows:

- **src/**
A folder where all your assembly files must directly reside in.
- **src/Makefile**
A makefile which can be used to create your sorting program. Specifically:
 - The default target must be made such that when executing “**make**” inside the **src/** folder, it must produce an executable “**sorter**”.
 - A target “**clean**” must exist that deletes all files that can be generated from the makefile. Typically that will be all object files and all executable that the makefile can generate.

Submission and Grading

Submission is done by uploading your zip-archive to itslearning. To allow time for focusing on report writing it is recommended that you are done with the programming and initial benchmark part by the end of week 42. The final deadline is

Sunday, 3 November, at 23:59 (CET)

Late submissions are not accepted.

This assignment will be evaluated together with your submission for the second assignment (for DM548) and the written test in January (both DM548 and DM588).

Bonus Challenges

There are many different sorting algorithms with different theoretical complexities, and multiple ways of implementing them. As a motivation we will have two competitions:

- Write the fastest sorting program (as determined by “**time**” on a Computer Lab machine) fulfilling the requirements above. This should be the default program (“**sorter**”).
- Write the smallest program, i.e., the one where the binary is the smallest. This program should be called “**smallSorter**”, and be buildable by invoking “**make smallSorter**” in the **src/** folder.

For these challenges you probably don't want to use the provided code snippets as they intentionally have not been written to be optimal. You are free to use SSE instructions and exploit the limited range of the numbers (they fit in 16 bit). You may also use additional system calls you find useful, as long as your solution is in line with the spirit of the project (if in doubt, ask).

When students are allowed to create their own groups, they can invite other students to work together on the assignment.

Until the student has created a group - or has chosen to work alone (submit individually) - it is not possible to answer the assignment.

The screenshot shows the 'Project plan' assignment page. At the top, there is a message: 'This is a group assignment. Please choose a group first!'. Below this, the 'Assignment' section states: 'Submit the project plan for the Christmas fair with your group.' On the right side, there is a sidebar with the following information: 'STATUS: Not submitted', 'GROUP ACTIVITY: Create your group' (with a green button), 'You can submit your answer individually' (with a blue link), and 'DEADLINE: 11/02/2018 23:59'.

To create a group, students type in the name of the student(s) they want to collaborate with.

The screenshot shows the 'Create your group' dialog box. It has a title bar with a close button (X). Inside, there is a text input field labeled 'Add: Type a name'. Below it, the 'Members:' section shows a list of members: 'Bloom, Peter'. At the bottom, there are two buttons: 'Save' (green) and 'Cancel' (white).

When saving the created group, invited members will get a notifications that they have been added to a group.

In the assignment itself, the group members are shown at the top of the page. It is now possible to start answering the assignment.

To add or remove members, a group member can click the Edit group button. This opens the same dialog as the one for creating the group. It is also possible to withdraw from a group.

Note that changing a group is currently only possible as long as there is no answer submitted.

The screenshot shows the 'Project plan' assignment page after a group has been created. The 'Group members' section at the top now lists three members: 'Bloom, Peter', 'Calabrese, Ronald', and 'Dutra, Erika'. The 'Assignment' section remains the same: 'Submit the project plan for the Christmas fair with your group.' and has a green 'Answer' button. The right sidebar is updated: 'STATUS: Not submitted', 'GROUP ACTIVITY: Edit group' (with a green button), 'Withdraw to join another group or submit individually' (with a blue link), and 'DEADLINE: 11/02/2018 23:59'.