Lab 1: Complexity Analysis

Martin Blom* and Jonatan Langlet†

Karlstad University, Sweden

November 10, 2020

1 Introduction

Suppose you are working as a full-stack developer at a local consultant company. An up-and-coming client recently launched the beta version of a brand-new service which—for production reasons—must be implemented in the `C` programming language without any external code dependencies. While the service works well for the most part, *some* beta users reported severe performance issues. After further investigation it appears that large service loads require better sorting and searching algorithms to scale better.

A reasonable first step is to understand the best, worst, and average case performance of the two algorithms that are used currently: bubble sort and

*martin.blom@kau.se
†jonatan.langlet@gmail.com

linear search. Other alternative algorithms should also be explored, such as insertion sort, quick sort, and binary search. It might be necessary to develop a stand-alone sorting and searching library, but for the time being a prototype with associated documentation and benchmarks suffices.

1.1 Preparation

Being a full-stack developer, it is important to master both front-end and back-end development. Review the material provided in the lectures and search online to deepen your understanding, reflecting over which type of coding would be categorized as front-end and back-end (respectively). If it helps, consider the start code from Section 2.1 as a concrete example.

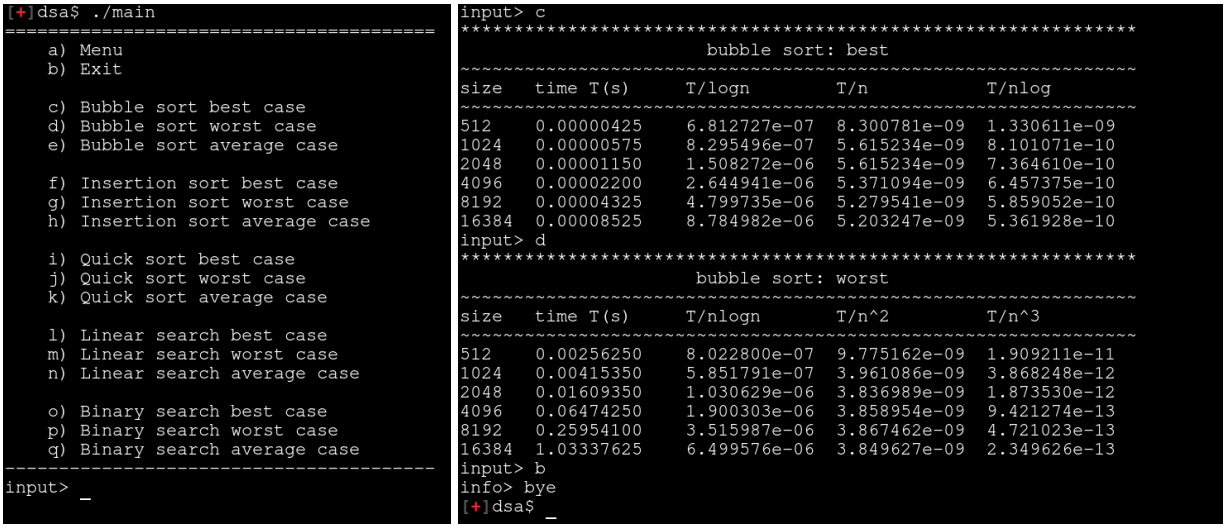
You should also freshen up your C-programming skills and learn how to measure time. See, for example, `man 3 clock` or `man 2 clock_gettime` [1]. Finally, study the following algorithms: bubble sort, insertion sort, quick sort, linear search, and binary search [3].

1.2 Learning Goals

This is an engineering project that requires initiative and originality. Apart from some general requirements and scoring criteria, you have a lot of freedom to design, implement, and write-up as you wish. Throughout the project you will deepen your understanding of `C`, abstractions, algorithms, performance benchmarks, and time complexities using big-$\mathcal{O}$ notation.

2 Task

Develop a menu-driven program that evaluates the best, worst, and average case performance of bubble sort, insertion sort, quick sort, linear search, and binary search. Your front-end should be similar to the one shown in Figure 1, invoking back-end functions to obtain performance results that can be formatted for viewing in table format. Each table must include the size of the underlying sequence being evaluated, real execution time, and a justification for the correct big-$\mathcal{O}$ complexity by looking for *convergence* when dividing by the right prediction [2]. For example, it appears that the best-case of bubble sort is $\mathcal{O}(n)$ because the time column divided by the size column converges to a constant. Similarly the worst-case appears to be $\mathcal{O}(n^2)$ because the time divided by *squared size* converges to a constant.



(a) Main menu (b) Bubble sort results

Figure 1: Example of a terminal-based front-end.

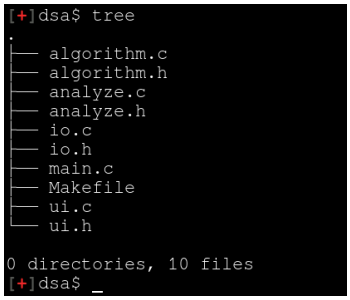


Figure 2: The `ui` and `io` modules compose a basic front-end, while the remaining modules (apart from `main.c`) implement a bare-bones back-end.

2.1 Getting Started

An overview of the start code is shown in Figure 2. While it is not a strict requirement to use it, your design must contain a clear distinction between front-end and back-end code. For example, it should be easy to add a different user interface *without* changing the back-end at a later stage. Regardless of if you build a new project from scratch or not, it must contain `algorithm.{c,h}` with the declared functions kept intact. In other words, you are not allowed to change the public interface of these algorithms.

The start-code is available on Canvas. Alternatively, you can download it from the terminal: `git clone https://git.cs.kau.se/rasmoste/dvgb03.git`. Note that we provided a Makefile that builds the entire project. If you are

using MacOS, you need to remove the linker flag `-lm` before compiling.

2.2 Points Awarded

You will receive at most eight points based on the submitted code and report, see Tables 1–2. While no points are awarded for good coding practises specifically, we reserve the right to give up to one penalty point for *excessive* use of global variables, too large functions, and redundant copy-paste programming. The listed report criteria are explained further in Section 3.

Table 1: Scoring criteria (code)

Task	Points	Criteria
Program design	1	Decoupled front-end/back-end
Quick sort	1	Correct implementation
Bubble sort	0.5	Correct implementation
Insertion sort	0.5	Correct implementation
Linear search	0.5	Correct implementation
Binary search	0.5	Correct implementation

Table 2: Scoring criteria (report)

Task	Points	Criteria
Algorithms	1	Principles, different cases
Results	1	Setup, clear presentation
Analysis	1	Execution time, big-$\mathcal{O}$
Completeness	1	Structure, language

3 Submission

Submit a report and all source code on Canvas as a zipped directory.[1] The deadline is **November 23 2020**. Remember to include `algorithm.{c,h}` with intact function declarations. The report may be in Swedish or English and should at least contain the following items:

- Name and email addresses of all group members (max two per group).

[1] Prepare a zipped directory from the terminal: `zip -r <name>.zip <path-to-directory>`

- Introduction that contains a summary of the lab's specification and necessary assumptions (if any).
- Background explaining preliminaries, such as the principles used by each algorithm and how to setup best, worst, and average cases.
- Justification of your design and implementation. What makes it solid?
- Experimental setup and results in an *easy-to-understand* format. Use figures and/or tables that are referenced in the text.
- Analysis comparing the different algorithms and their big-$\mathcal{O}$ complexities. Discuss your results—both the good, the bad and the ugly!
- Encountered problems (if any) and how you solved them.
- Concluding remarks: what is the take away and what did you learn?

Acknowledgements

This lab was initially created by Donald F. Ross. The current start-code and specification was repackaged by Rasmus Dahlberg and Martin Blom.

References

[1] Elapsed time. https://web.archive.org/web/20181228072206/https://www.gnu.org/software/libc/manual/html_node/Elapsed-Time.html. Accessed 2019-04-14.

[2] Weiss ex 2.4.5. https://web.archive.org/web/20190414163607/https://www.cs.kau.se/cs/education/courses/dvgb03/lab_info/index.php?Weiss=1. Accessed 2019-04-14.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.