Halmstad University course DA4002
Introduction to Algorithms, Data Structures, and Problem Solving

Assignment for Project 1: Sorting Algorithm Benchmarks

Roland Philippsen
roland.philippsen@hh.se

September 25, 2013

Introduction

Complexity analysis is an important tool to evaluate possible solutions for a given problem. Lecture 4 and the course book by Loudon [1] discuss several theoretical aspects of computational complexity, with a focus on how runtime grows with problem size.

The first project complements the theoretical treatment with hands-on experience. It provides a practical and empirical understanding of the theory, and prepares the participants to make informed choices in real-life programming situations.

The deadline for handing in the source code and the report is **Friday, October 4, 2013, at 18h00**. Teams who miss the deadline will receive a penalty (redcued grade). But of course, in case of exonerating circumstances such as sickness, a deadline extension will be granted. Please inform the lecturer as soon as possible when something like this happens.

Participants are provided with fully functional (but limited) benchmarking programs for sorting algorithms which determine how long it takes to sort arrays of varying lengths. The two provided files `insertion-sort-benchmark.c` and `merge-sort-benchmark.c` use arrays of random integers as input data, measure the time it takes to sort them, and prints the data in a format that can be used for plotting with `gnuplot`. This is very similar to what is done in exercise 7. Several plotting examples are provided with the benchmark sources.

There is a set of mandatory tasks that have to be performed in order to pass. And there is a list of suggested bonus tasks which can lead to a higher grade. A properly performed set of mandatory tasks that are well documented in the project report will receive a grade of 4 in the Halmstad system (ECTS grade C). By implementing bonus tasks and properly documenting them, the highest grade (5 and A) can be reached. Note that the projects, as well as the written final exam, each count independently for 2.5 credits. Each of these must be passed individually, and the final overall grade is the average of all three.

The report is an integral part of the project and counts for approximately half the points. In the report, it has to become clear what the students have done, where they took the relevant information, and to what extent the different questions can be answered. Remaining open issues should also clearly be identified. An example report is provided, and this template should be followed rather closely.

The internet is an important resource for real-world programming. Using information from the internet is strongly encouraged, but it has to be clearly stated which sources were used for the various parts of the work. For example, URLs for algorithm pseudo code or complexity estimates should be explicitly stated in the project report. Wikipedia articles are provided as starting points.

During three regularly scheduled lab sessions, students can ask for help working on their project in the Unix room B231c (the same as the exercise room). The lecturer and course assistant will help with any question that may arise (short of implementing the project for

algorithm	difficulty	link
cocktail sort	easy	`http://en.wikipedia.org/wiki/Cocktail_sort`
selection sort	easy	`http://en.wikipedia.org/wiki/Selection_sort`
gnome sort	easy	`http://en.wikipedia.org/wiki/Gnome_sort`
radix sort	easy	`http://en.wikipedia.org/wiki/Radix_sort`
shell sort	moderate	`http://en.wikipedia.org/wiki/Shell_sort`
comb sort	moderate	`http://en.wikipedia.org/wiki/Comb_sort`
cycle sort	moderate	`http://en.wikipedia.org/wiki/Cycle_sort`
quicksort	hard	`http://en.wikipedia.org/wiki/Quicksort`
heapsort	hard	`http://en.wikipedia.org/wiki/Heapsort`

Table 1: List of sorting algorithms and their difficulty level.

the students) without any influence on the final evaluation. Teams are encouraged to talk with each other in order to figure out common programming issues, but copying source from each other code for sorting algorithms or the core benchmarking setup is not allowed.

Assignment

Table 1 lists sorting algorithms which can be added to the benchmark. The (estimated) difficulty of implementing each algorithm is also given, along with a link to a Wikipedia page providing more details. Note that the list excludes bubble sort, insertion sort, and merge sort, because they are either provided as part of the project starting point or have been treated during an earlier exercise.

Your code must compile and run properly on the Unix machines in room B231c. You can develop it on some other machine, even another operating system, as long as you make sure that it works correctly on the University's Unix workstations.

Mandatory Tasks

1. Browse the Wikipedia pages listed in table 1. Then, choose **two *easy* and one *moderate*** algorithms, and create benchmark programs for them.

   - Copying and adapting one of the provided source codes is probably the easiest way to achieve this. Please choose useful file names for the new benchmarks, and also useful function names, so that it becomes clear what algorithms have been chosen and where to find their implementation. Also, make sure adapt things like the messages printed by the benchmark, to reflect the specific algorithm.
   - Add code that verifies the correctness of your implementation. This could be a separate program, or inside the benchmark.
   - Produce plots which clearly show the running times of the added algorithms, both individually and in relation to each other (including insertion sort and merge sort). You will probably need to adjust the variables `nstart` and `nmax` in order to get good coverage of the relevant array lengths, without creating a benchmark that takes too long to run.

2. Reduce the effect of variation due to noise in the runtime measurements. This should be done by measuring each data point 10 times (with different random input arrays) and computing then minimum, maximum, and average runtime for each N. Create plots which clearly illustrate the benefit of such repeated measurements.

3. Determine how well the measured runtimes match the theoretical Big-Oh complexity classes for all of the sorting algorithms available to you. This includes insertion sort

and merge sort, and if you do the bonus task on *hard* algorithms then also include those. Create corresponding plots and discuss your findings in the report.

Note that, if you get really noisy runtime measurements, it may be best to use the minimum instead of the average measured runtime.

Bonus Tasks

Perform one or more of these suggested tasks to be able to receive the highest grades.

1. Perform the mandatory tasks on two additional *moderate* algorithms from table 1.
2. Perform the mandatory tasks on one additional *hard* algorithm from table 1.
3. Investigate the dependency of the runtime measurements on input data distribution. Do this by also measuring the time it takes each algorithm to process data which is already sorted, sorted in reverse, and partially sorted. Create clear plots which illustrate the differences between the sorting algorithms for all these three kinds of input. Clearly discuss in the report which algorithms show a dependency on the input data order, and what the observed effects are.

Further Information

Plotting Scripts

Gnuplot can be used interactively (like in exercise 7) by just giving its command in a terminal. But it can also run in batch mode by passing the name of a plot script file on its command line. A gnuplot script file is just a text file with a list of gnuplot commands. You can download an archive with several example scripts from the course website. They have a "`.plot`" extension on the file name. Notice that those examples which create a figure window must be launched with the additional option "`-persist`" as otherwise the figure just flashes briefly on the screen. There is one example plot file which save the figure in a PNG image, intended for inclusion in your report. Please look at those files and read the gnuplot documentation [2] for more details.

You are free to use some other software to plot your data if you are having trouble with Gnuplot. For example, if you are familiar with Matlab, you may find that easier to work with. But you may have to change the output format of the benchmark program accordingly. Excel or LibreOffice Calc may be other viable options. Note that whichever option you choose, your report must document the plotting tool and commands such that your figures can be reproduced.

Submission Format

Send your report in PDF format and all your source files packaged into a tar or zip archive. Make sure the files you placed in the archive are complete and free of errors. The name of the archive and the report must contain the last names of all team members. The report must also clearly state the names of the team members. Please do not use spaces in file names.

References

[1] Kyle Loudon. *Mastering Algorithms with C.* O'Reilly Media, 1999.

[2] Thomas Williams, Colin Kelley, and many others. Gnuplot. `http://www.gnuplot.info/`, September 2011.