Introduction to Algorithms, Data Structures, and Problem Solving

DA4002 (HT11) Halmstad University

Example Report with Guidelines
Project 1: Computational Complexity

Roland Philippsen
course examiner
roland.philippsen@hh.se

September 21, 2011

1 Introduction

A (short!) written report is an essential part of the project. It must clearly identify the team and present the work performed by them. This example report provides a recommended format.

The introduction is the place to introduce the problem and briefly state what results have been achieved. For project 1, this means stating which benchmark problem you have chosen to solve. You should also briefly explain how far you have advanced through the mandatory tasks (this can be as simple as saying "all mandatory tasks have been done"), and whether you have done any bonus tasks.

In case you hand in a partial solution, this should already be mentioned in the introduction, but details should wait until the implementation or results section. If you need rather long explanations for some of these aspects, you can put that into a separate discussion section.

This example report uses the two benchmark applications given to the course participants as technical examples. It presents how to run the benchmarks and the basic plots that can be produced.

2 Implementation

The implementation section should introduce any new sources, classes, and methods. You should also explain how you have integrated your changes into the benchmark.

For example, if you have chosen alternative B and implemented *bubble sort* [2], you could write something like this: "The source file `BubbleSort.java` contains a class called `BubbleSort` which has a method called `sort` that sorts a given array of strings using the bubble sort algorithm. The implementation is based on the pseudo-code shown in figure 1."

As you can see, it is acceptable to copy-paste *short* sections from other sources, such as websites or books, *as long as* this is properly marked as a quotation and the original source is given. Note that bubble sort is *not* on the list of algorithms you can choose for the mandatory tasks of alternative B, because bubble sort has already been done during exercise 3.5.

The two example benchmark applications that are provided as starting points for the first ITADS project are partially based on source code that was already seen during exercises. In general, source file names match class names, so `ExampleA.java` contains class `ExampleA` which has a `main` method that measures the runtime of inserting items into a list and a vector. `ExampleB.java` is very similar, except that it measures the time it takes to sort arrays of random strings with the merge and insertion sort algorithms. Both these examples rely on utilities and other classes implemented in the other files that come bundled in the source archive for project 1:

- `Factory.java` contains a class which provides methods for generating input data. Most importantly, the `createRandomStrings` method creates an array of N strings. Each string in

```
procedure bubbleSort( A : list of sortable items )
   n = length(A)
   repeat
      newn = 0
      for i = 1 to n-1 inclusive do
         if A[i-1] > A[i] then
            swap(A[i-1], A[i])
            newn = i
         end if
      end for
      n = newn
   until n = 0
end procedure
```

Figure 1: Pseudo-code for bubble sort, taken from Wikipedia [2].

that array will be between 2 and 11 characters long, and it is composed of random alphanumeric characters. The other methods are `createSamples` (which is useful for some of the bonus tasks), various `duplicate` methods, and a short `main` which serves as demo.

- `LogEntry.java`, `LogSeries.java`, and `LogBook.java` contain classes that log the benchmark data and produce the files and plot scripts that can be used to create figures for presenting your results. The most important methods are the following:
  - `LogBook.addSeries` defines a new `LogSeries` instance to be added to the log.
  - `LogBook.createFiles` writes all data of all registered `LogSeries` objects, and also creates plot script files that reference these log data files.
  - `LogSeries.startSingle` and `LogSeries.stopSingle` are used to record a single data point.

  The `main` methods of the `ExampleA` and `ExampleB` classes clearly illustrate how to use the logging facilities.

- `StringList.java`, `StringListIterator.java`, `StringListNode.java`, and `StringVector.java` contain classes which implement the doubly-linked list and the dynamically growing array that have been developed during exercise 2.

- `InsertionSort.java` and `MergeSort.java` contain the classes which implement the sorting algorithms used by `ExampleB`. Both of them contain a `sort` method which applies insertion sort [3] and merge sort [4], respectively, on an array of strings.

Another important thing to explain in the implementation section is how to compile and run the applications that you have written. The code provided as starting point of the project is compiled using the `make` command, which invokes GNU Make [1]. This will read build instructions from the `Makefile` which is part of the sources, and create `.class` files for all the `.java` files in the directory.

Running the example applications is simply a matter of passing their class names to the `java` virtual machine. This is done by opening a terminal and giving the following commands:

```
java ExampleA
java ExampleB
```

Running `ExampleA` produces console output like the one in figure 2. And `ExampleB` is very similar, as can be seen in figure 3.

In case you hand in some partially solved tasks which do not compile, this should be clearly explained: which file does not compile, and why? What would need to be done in order to make it compile? Any compilation errors that are not covered by such an explanation are a serious failure.

```
time required for container operations
--------------------------------------
N        list push front        vector push back
1024                2                0
1448                2                0
2048                1                0
2896                1                1
4096                4                0
5793                0                1
8193                1                1
11587               1                2
16386               5                1
23173               2                1
32772               6                2
46347               5                3
65545               6                13
92695               3                4

to view all data in one figure on screen, run this command:
    gnuplot -p log-1316350931155-all-scr.plot

to view separate figures on screen, run this command:
    gnuplot -p log-1316350931155-sep-scr.plot

to create a PDF figure with all data, run this command:
    gnuplot log-1316350931155-all-pdf.plot

to create separate PDF figures, run this command:
    gnuplot log-1316350931155-sep-pdf.plot

to view the PDFs, you can use e.g.:
    evince log-1316350931155-*.pdf &
```

Figure 2: Typical console output from `ExampleA`. The table with the running times gives feedback while the benchmark is being run. The list of commands at the end allows to view and create figures.

```
time required to sort data
--------------------------
N        insertion sort        merge sort
256                4                1
362                2                1
512                1                7
724                2                0
1024               2                1
1448               6                0
2048               8                1
2896               17               1
4096               32               1
5793               64               2
8193               129              2
11587              265              4
16386              559              5
23173              1199             8
32772              4245             15

to view all data in one figure on screen, run this command:
    gnuplot -p log-1316350944986-all-scr.plot

to view separate figures on screen, run this command:
    gnuplot -p log-1316350944986-sep-scr.plot

to create a PDF figure with all data, run this command:
    gnuplot log-1316350944986-all-pdf.plot

to create separate PDF figures, run this command:
    gnuplot log-1316350944986-sep-pdf.plot

to view the PDFs, you can use e.g.:
    evince log-1316350944986-*.pdf &
```

Figure 3: Typical console output from `ExampleB`. Similarly to figure 2, the table is produced while the benchmark is being run. Again, the list of commands at the end allows to view and create figures.

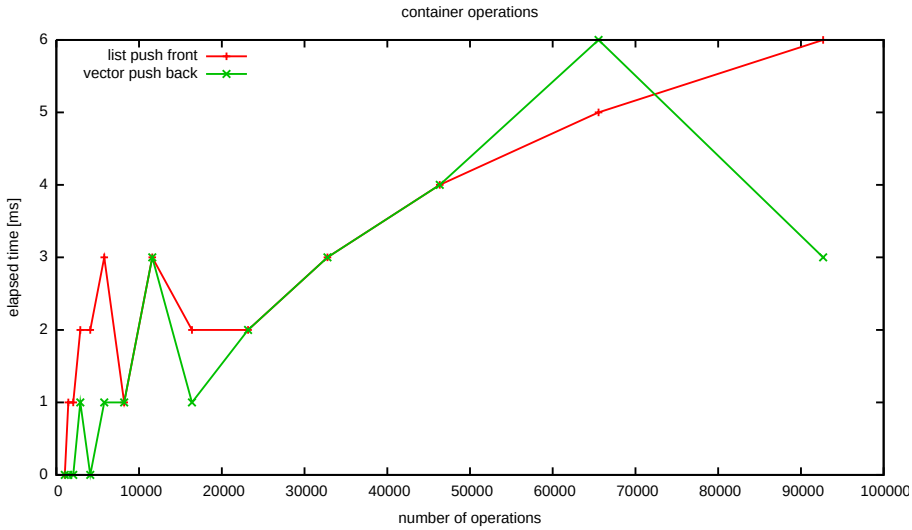

Figure 4: Execution time plot produced by running example A, and then using the automatically generated plot script like this: "`gnuplot log-1316350931155-all-pdf.plot`"

In a more general sense, when writing applications that must be used by others (who may not be programmers, or who may not have access to the source code), the explanations you write here are the only way for people to find out how to use your application. For this project, it is enough to give the exact command required to produce each figure in the results section, and you can do that inside the results section. If your code produces some special output on the terminal which is needed to understand your results, then you should also include some example output and explain which part is special, and why.

3 Results

In the results section, you have to present the obtained technical results. For the first project, this means that for each task, you have to produce one or more figures which clearly illustrate the point you are making. For example, if you found out that the running time of a sorting algorithm gets much worse if the input data is in reverse-sorted order, then you have to include a figure which shows just this effect. It is also important to summarize your main findings in order to give a high-level understanding. This can be done at the end of the results section, or in a separate discussion or conclusion. Sometimes, it also makes sense to additionally (and very briefly) mention your main result already in the introduction.

The two example applications produce data and plot files that are ready to use. `ExampleA` produces plots such as the one shown in figure 4. This figure shows that the insertion times for lists and vectors are quite short. It also shows that the variation in the measurement can be quite high. This makes it hard to see that the running time actually follows an $O(N)$ complexity curve. After you have performed the third mandatory task, these plots should produce somewhat smoother curves.

`ExampleB` measures the running time of insertion and merge sort. As shown in figure 5, there is a significant difference between these two algorithms. Insertion sort is known to be $O(N^2)$ [3] and this fits qualitatively. Of course, a proper empirical analysis would have to include a more precise technical demonstration of whether the measured running times actually follow a quadratic curve. This was presented during lecture 4, and is one of the bonus tasks available for alternative B.

Another important point visible in figure 5 is that merge sort is much faster than insertion sort. So, figure 6 shows just the data of merge sort. From that figure it would be hard to see that merge sort is $O(N \log N)$ [4]. One reason is that the benchmark was not pursued for large enough problem sizes, and another one is the "noise" for very small problems. These kinds of issues can be addressed in a bonus task. For instance by creating a separate benchmark which looks just at

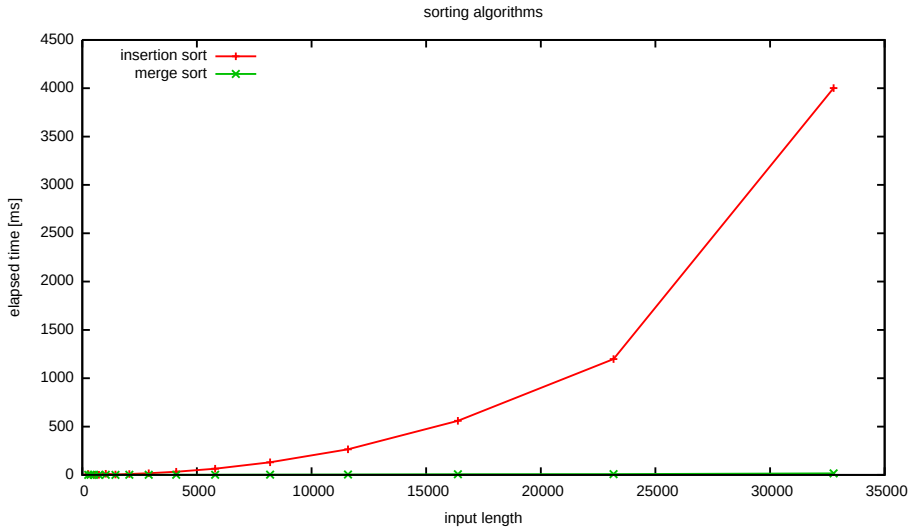

Figure 5: Execution time plot produced by running example B. Merge sort is so fast that its running time graph almost coincides with the X axis.

merge sort (and maybe other $O(N \log N)$ sorting algorithms) over a larger range of problem sizes, and matching it with the theoretical growth curves.

Summarizing the results of running the example benchmarks, it can be said that they give a good starting point for further work. The data they produce is a bit too noisy, and they do not compare enough different algorithms and data structures to make really general statements. But if they did that, then the course participants would have nothing to do during their projects...

4 Further Remarks

As explained during lecture 4, the submission format for the projects is as follows:
– Submit either individual files, or a ZIP or TAR archive.
– All Java source code must be handed in.
– The report must be in PDF format.
– Everything has to be sent via email to the course examiner by Friday, September 30, at 18h.

The ITADS course focuses on technical aspects of programming and problem solving. The quality of the written language in the report is secondary. As long as it remains comprehensible, English errors are not relevant. Verbatim quotes (copy-pasting) from other sources is acceptable **if they are short, properly marked, and adequately cited**.

If language is a major obstacle for a team, it is possible to receive a deadline extension for the report only. Source code will still have to be handed in at the normal deadline. Notify the lecturer as early as possible in order to arrange a deadline extension for the report.

Halmstad University offers an **English Language Studio** for students who seek help with written English. The studio is available on Wednesdays between 9:00 and 12:00 in room F-208. The teacher is Nicholas Lloyd-Pugh `<nicholas.lloyd-pugh@hh.se>` (phone 167372).

5 Conclusion

The most important points to remember about the report are:

- Mention course information: University name; course name, code, and date.
- Mention team information: names, email addresses, and study programme.

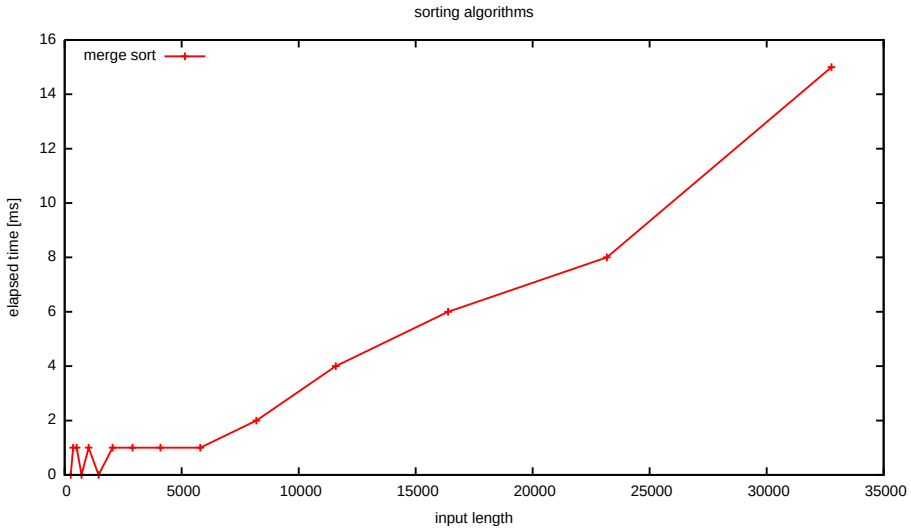

Figure 6: Execution time plot of merge sort.

- Don't forget the project title and subtitle.
- The introduction section must summarize the work you have done.
- The implementation must present what you have added and changed, and how to run it
- the results must clearly show what you have achieved
- a separate discussion section is possible if you want to give a high-level explanation of the detailed technical results
- a conclusion is also a nice additional section, it allows you to make some overall comments and maybe point to future work
- references must be given to all the web sites, books, and other material that you have used[1]; it is acceptable to quote (copy-paste) those sources, as long as the quotes are short, clearly marked, and properly referenced

This example report mixes technical aspects and general writing guidelines. Your project report will probably be much shorter, which is just fine (and less work for the evaluation). Good luck with your work!

References

[1] Free Software Foundation. GNU make. `http://www.gnu.org/software/make/`, September 2011.

[2] Wikipedia. Bubble sort. `http://en.wikipedia.org/wiki/Bubble_sort`, September 2011.

[3] Wikipedia. Insertion sort. `http://en.wikipedia.org/wiki/Insertion_sort`, September 2011.

[4] Wikipedia. Merge sort. `http://en.wikipedia.org/wiki/Merge_sort`, September 2011.

[1] except the most obvious sources, such as the course book or the lecture slides