My first report

My Name

Spring Term 2023

Introduction

This is what a report should look like. It is written using the document class `article` with `a4paper` and `11pt` options. You should of course replace the "My first report" with something more descriptive and of course have your name below the title.

What follows is a set of rules and hints on how to write you reports. Follow these guidelines to make life easier and avoid failing an assignment by including a screen shot. Do read these guidelines but also look at the source code of this document. The code will hopefully show you how to do things. It will also show what packages are included to get things to work.

Page layout

Use vanilla LaTeXwith regular page width and height and single spaced lines. Don't use any fancy packages that will turn your report into a Christmas tree, keep it simple!

Since this is a small report you can omit having numbered sections and you do this by using section commands that end with a `*` (for example `\section*`). You can of course have subsections etc but if you don't have numbers on the main sections don't add numbers to the subsections.

Inserting code

Code snippets are included using the package `minted`. To get it to work on you laptop you need to have Pygments installed. If you use it in Overleaf it should work directly.

```
def append([], b) do b end
def append([h|t], b) do
  [h | append(t, b)]
end
```

If you want to include a program statement in running text you can do this using for example teletype-text: `append([1,2,3],[4,5])`.

The reports that you hand in should be up to four pages long - but not four pages of code! Use code snippets where you want to describe how things are done but don't include code just because you have written it.

Numbers

You will include some run-time measurements in your reports. You should then think about the number of significant figures that you use. Just because a benchmark took $12345678\mu s$ does not mean that you should report it in this way. If you write this in your report you're implicitly saying - if I do this again the run-time will be the same. This could be true but I doubt that anything you do on a computer can be determined with an 8 figure accuracy. The next time you try it might very well take $12354678s$. What you report is maybe $12.3s$ or $12s$?

Do read the paragraph above one more time. Handing in a report where time measurements are reported with more than three figures accuracy is a sure way to fail - since I then know that you have not read the above paragraph twice.

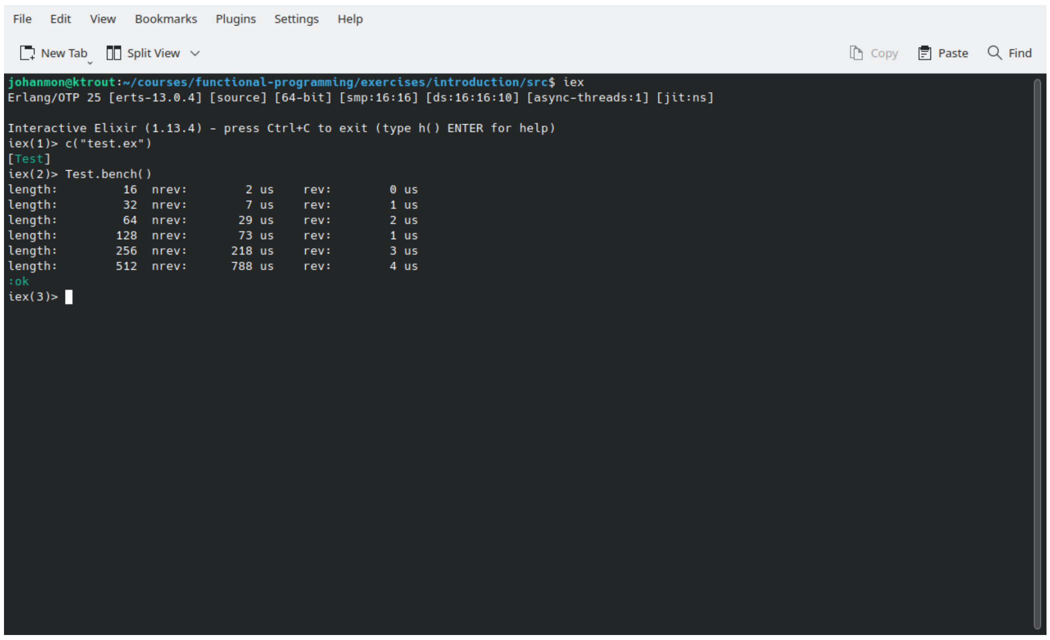
Tables

Numbers are often best presented in a table. You will have to do some reading on how to format tables but the general structures is quite easy. This is for example a table with some run-time figures.

prgm	**run time**	**ratio**
dummy	115	1.0
union	535	4.6
tailr	420	3.6

Table 1: Union and friends, list of 50000 elements, run time in micro seconds

As you see in the table above, the run time per se might not be interesting. The interesting thing is how it relates to something else. Look at the ratios above, it gives you the information that we are looking for. So when you include numbers, ask yourself why you have these numbers in the report. What is the purpose, can you describe it in a better way?



```
johanmon@ktrout:~/courses/functional-programming/exercises/introduction/src$ iex
Erlang/OTP 25 [erts-13.0.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [jit:ns]

Interactive Elixir (1.13.4) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> c("test.ex")
[Test]
iex(2)> Test.bench()
length:         16  nrev:        2 us    rev:        0 us
length:         32  nrev:        7 us    rev:        1 us
length:         64  nrev:       29 us    rev:        2 us
length:        128  nrev:       73 us    rev:        1 us
length:        256  nrev:      218 us    rev:        3 us
length:        512  nrev:      788 us    rev:        4 us
:ok
iex(3)>
```

Figure 1: This should never be used

No f*ing screen shots

I know that you are all very happy that things actually work and eagerly want to show what things look like on you screen but please, don't use *screen shots*. It looks ugly and it's impossible to mark or copy the things that you want to show. It also, most often, show a lot of irrelevant things so instead of using an image, copy the text and format it so it's easy to read.

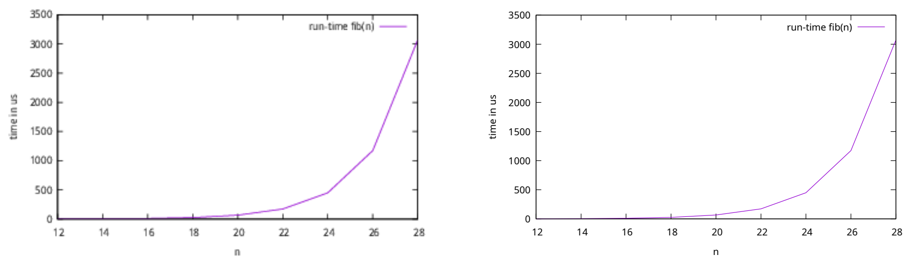
Graphs

Once you start to generate graphs make sure that they are readable and have sensible information on the axes.

There are many ways to generate graphs but you want to use a way that minimize manual work. My tool over the years has been *Gnuplot* and if you do not have a favorite tool you could give it a try. The Gnuplot program is a stand alone program that will generate a graph that you then can include in you report.

If you work with Gnuplot you should write the commands needed to generate a diagram in a small script. Take a look in the file `fib.p` and you will see how the diagram in Fig.2 was created from the data given in `fib.dat` (the `.png` file was generated from `fib.pdf` using the Linux `convert` program found in `imagemagick`).

When you include graphs you should make sure that the images you include are not raster images (gif, png etc) but a vector image that scales when you zoom-in. In Fig.2 you see the same graph saved as a raster image (png) compared to a vector graphic image. You might not see the difference but if you zoom-in you will see that the vector image scales.



(a) using raster graphics (b) using vector graphics.

Figure 2: Difference in image formats.

An alternative to including a graph produced by a separate program is to describe the graph in LaTeX. This can be done using the TikZ library. This library is used to create all types of graphics and the learning curve is quite steep. The benefit is that the LaTeXdocument becomes self contained and that you are in complete control over the result.

The data can either be written in the latex source file but better read from a separate file. Reading from a separate file makes it easier to combine the output from a benchmark with the report. If you construct your benchmark to produce a file with the x and y values in columns you can plot them using a simple `\addplot` command. If you do changes to your program you simply run the benchmark again and re-compile the LaTeXfile.

The graph in Fig.3 is generated using Tikz and as you can see, I know have the time in "μs" instead of in "us".

Errors

Some LaTeXerrors that I frequently see that could easily be avoided if you only know where they come from.

less than

If you in your LaTeX code write "$5 < 7$" it will look like 5 ¡ 7 and "$9 > 7$" will look like 9 ¿ 7. Using the characters $<$ and $>$ directly does not work ... so, how did I do it? I used the commands `\textless` and `\textgreater` to generate the symbols $<$ and $>$.

You could also use `{\tt 5 < 7}` but then it will use the teletype font and look like this: `5 < 7`.

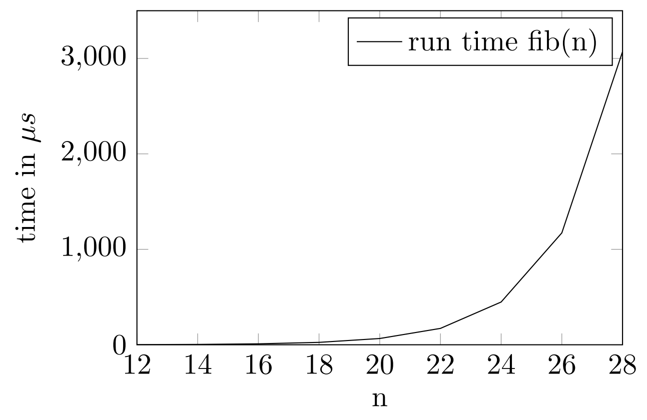


Figure 3: The same graph using TikZ

Still another way is to write it using so called `math mode`. This is a mode used for writing mathematical formulas in a nice way. You enclose your expression in `$` signs like this `$5 < 7$` and then it will look like this $5 < 7$.

If you have a larger mathematical expression you enclose it in double $ and the result is that it is written centered with some space around it like this:

$$5 < (3 * 8/3)$$

why strange font

If you want to write *foo* in teletype font you write like this `{\tt foo}`. If you forget the closing } then it will look like this: `foo. Now everything here after until the end of you report will look like this.`

Make

To automate a process of running benchmarks and compiling a report one can add everything that needs to be done using a `Makefile`. The `make` program will determine which files that needs to be regenerated and re-do only the necessary steps. If you take a look at the make-file that comes with this report you will see that a change to the Fibonacci benchmark (`fib.ex`) will trigger the file `fib.dat` to be regenerated. This will in turn mean that he diagrams are regenerated and in the end the LaTeXreport i recompiled.

Working with make-files that causes more than just the report to be regenerated in one reason why it's more powerful to run LaTeXon your own machine rather than using *Overleaf*. It does require some tinkering to get everything to work but once you have it up and running the development cycle becomes much shorter.