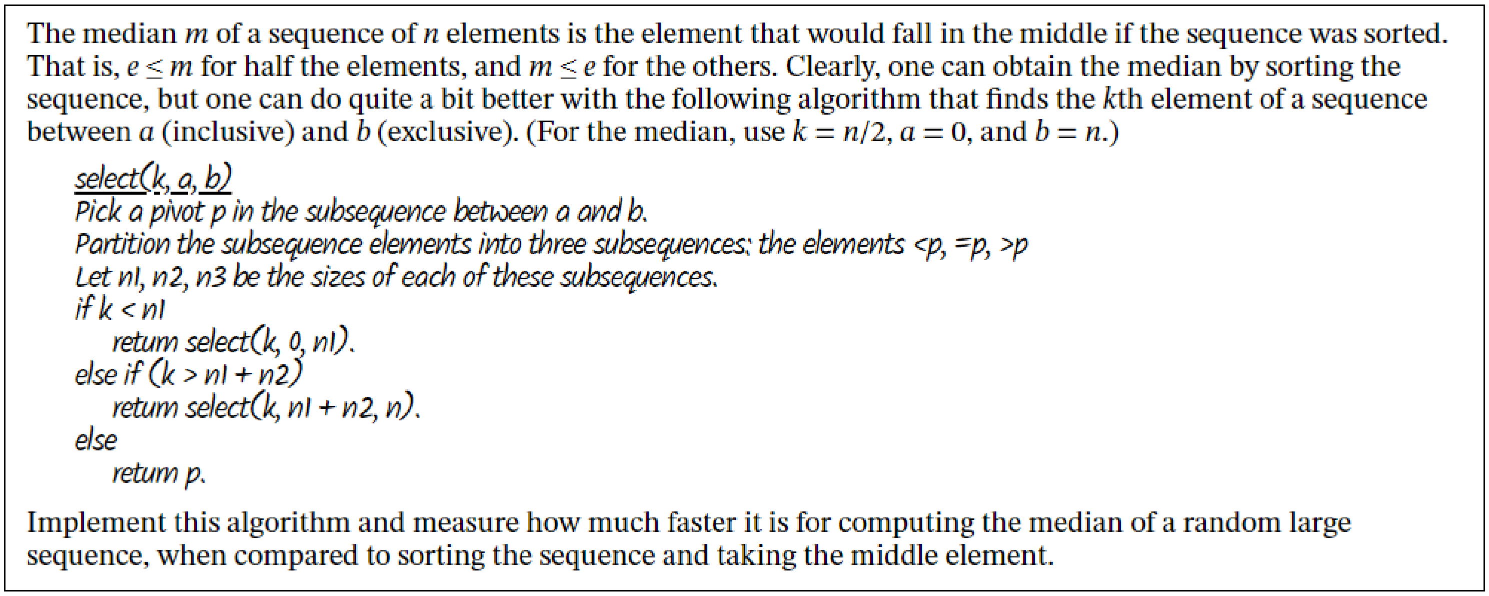
Lab 3

Due Mar 9 by 8am **Points** 10 **Submitting** a file upload **File Types** pdf and tgz
Available until Mar 9 at 8am

This assignment was locked Mar 9 at 8am.

For this lab we are comparing the timing of *three* algorithms. Typically, to find a *median*, we would *sort* the data, then select the middle element.

Implement this first using the built in **sort** algorithm, then using **quicksort**, then compare the time those take in milliseconds with this idea for **select**:


The median m of a sequence of n elements is the element that would fall in the middle if the sequence was sorted. That is, $e \leq m$ for half the elements, and $m \leq e$ for the others. Clearly, one can obtain the median by sorting the sequence, but one can do quite a bit better with the following algorithm that finds the kth element of a sequence between a (inclusive) and b (exclusive). (For the median, use $k = n/2$, $a = 0$, and $b = n$.)

```
select(k, a, b)
Pick a pivot p in the subsequence between a and b.
Partition the subsequence elements into three subsequences: the elements <p, =p, >p
Let n1, n2, n3 be the sizes of each of these subsequences.
if k < n1
    return select(k, 0, n1).
else if (k > n1 + n2)
    return select(k, n1 + n2, n).
else
    return p.
```

Implement this algorithm and measure how much faster it is for computing the median of a random large sequence, when compared to sorting the sequence and taking the middle element.

Run your program using a small array to test that each algorithm works using ***Catch2*** unit tests.

Then use a ***very large array*** of numbers until it takes at least some milliseconds. Otherwise you can't see any difference in the timings!

Run your program with the options

```
-s -d yes -r compact -o results
```

to capture the output into the results text file and include it using the **VerbatimInput** command.

Add comments to the ***select*** function of your program explaining what the function does and how it manages to find the median. The idea is to help the reader understand the code so do not simply paraphrase the code, but instead add ***explanations with illustrations and/or drawings*** clarifying the change of state that the program produces.

If you use **ddd** to create the illustrations, you can save them as .ps files, then use the convert command:

```
mogrify -format png *.ps
```

to create images that can be included in the pdf file.

If you use gdb from console, you can

```
set logging file gdboutput

set logging overwrite on

set logging on

show logging
```

then include gdboutput using **\VerbatimInput.**

or you could use the verbatim environment to enclose outputs that you pasted from one of these log files.

\begin{verbatim}

....

\end{verbatim}

In either case, you still must annotate or somehow explain the meaning of the data.

Document readers are not mind-readers!

Add a summary page where you comment about the differences in timings between the 3 algorithms.

Use **cpp2pdf** to create the PDF that will have all source, code, images and comments nicely formatted and easy to read. Use the Latex **\newpage** command where needed to make logical breaks in the program structure.

Save your work by using this command in the working directory (lab3), retrieve the tgz and pdf file using the web server, and upload and submit to Canvas.