Lab 2

Due Feb 23 by 8am **Points** 10 **Submitting** a file upload **File Types** pdf and tgz
Available until Feb 23 at 8am

This assignment was locked Feb 23 at 8am.

For this lab we are comparing the timing of two algorithms. Typically, to find a *median*, we would *sort* the data, then take the middle element if the number of elements (n) are odd, and the average of the two middle elements if n is even.

Implement this first using the built in **sort** algorithm, then compare the time that takes in milliseconds with this idea (we are developing the **partition** function together in class)

> 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.

If your program takes less than a millisecond, then use a larger array of numbers until it does take at least a millisecond. Otherwise you can't see any difference in the timings!

e.g. In Linux shell, you can generate numbers like this:

```
seq 10 | shuf
```

Use **Catch2** (https://github.com/catchorg/Catch2/blob/master/docs/tutorial.md) to unit test the results of each of these two algorithms.

Run your program with the options ***-s and -d yes*** and capture the output into a png file. Then include that image in block comments of your program using the Latex **includegraphics** command. Or redirect the output into a text file and include it using the **VerbatimInput** command. Add comments to each function of your program explaining what the function does. 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.

Use **cpp2pdf** to create the PDF that will have all source, code, images and comments nicely formatted and easy to read. Use the Latex newline command where needed to make logical breaks in the program structure. Change author to your name, not mine.

Save your work by using this command in the working directory (lab2), retrieve the tgz and pdf file using the web server, and upload and submit to Canvas.

```
save . *.pdf
```