1.2 Report: You must submit an assignment report in pdf format. Your clear and concise report should contain the following:

* An Introduction, comprising a short description of the algorithm and why it is worth considering parallelization.
* A Methods section, giving a description of your approach to the solution, with details on the parallelization. This section must explain how you validated your algorithm (showed that it was correct), as well as how you timed your algorithms with different input, how you measured speedup, the machine architectures you tested the code on and interesting problems/difficulties you encountered.
* A Results and Discussion section, demonstrating the effect of data sizes, filter sizes and sequential limits on parallel speedup. This section should include speedup graphs and a discussion. In this section, we expect you to should answer the following questions:
* Is it worth using parallelization (multithreading) to tackle this problem in Java?
* For what range of data set sizes and filter sizes does your parallel program perform well?
* What is the maximum speedup obtainable with your parallel approach? How close is this speedup to the ideal expected?
* What is an optimal sequential cut-off for this problem? (Note that the optimal sequential cut-off can vary on dataset size and filter size.)
* A Conclusions section (note the plural) listing the conclusions that you have drawn from this project. What do your results tell you and how significant or reliable are they?

# Introduction and Theory

Median filtering is a nonlinear digital filtering technique that is often used to remove noise from a data set. In this method, the median filter slides over the data sequentially and replacing each value with the median of its neighbouring entries (the number of which is specified by the filter size). In this assignment, I am going to investigate the effect of parallelising the algorithm used to filter the noise out of the data set. The basic algorithm to filter the result set is as follows (in pseudo code):

*//loop through the data set and apply the filter to each element*

**for** each element in dataset{

element = calculateMedianOf(element and its neighbours);

*//the number of neighbours to be used depends on filter size*

}

The problem with this naïve approach is that, because the size of data set is exceptionally large, going through the data set sequentially will take a very long time since the algorithm’s runtime is ***O(n)*** and ***n*** is too large to be ignored in this case.

To improve the speed of this algorithm, I am going to parallelise it using the *Java Fork/Join Framework* to give an ***O(log(n))*** runtime*.* Parallelising will allow me to process different parts of the data set at the same time so that the whole data set is filtered faster. I am going to compare the performance of the sequential approach and the parallel approach, using different filter sizes, data set sizes and also on different computer architectures in an effort to produce the most reliable results and conclusive conclusion to the research.

My predictions for this research are:

* The parallelised program should be much faster than the sequential one for the same filter size and computer architecture.
* The speed of both approaches with small data sizes should not differ by a big margin since *Big O* analysis is not reliable for small “*n*”.
* The program’s performance should decrease with increasing filter sizes.

# Methods

* Create sequential program.
* Test running speed with different architectures, data set sizes and filter sizes
* Create parallel program
* Make sure it produces same results as sequential
* Test running speed with same cases as sequential

Results and Discussions

* Compare the set of results (sequential vs parallel) using graphs and other relevant methods
* Verify predictions
* Conclusions