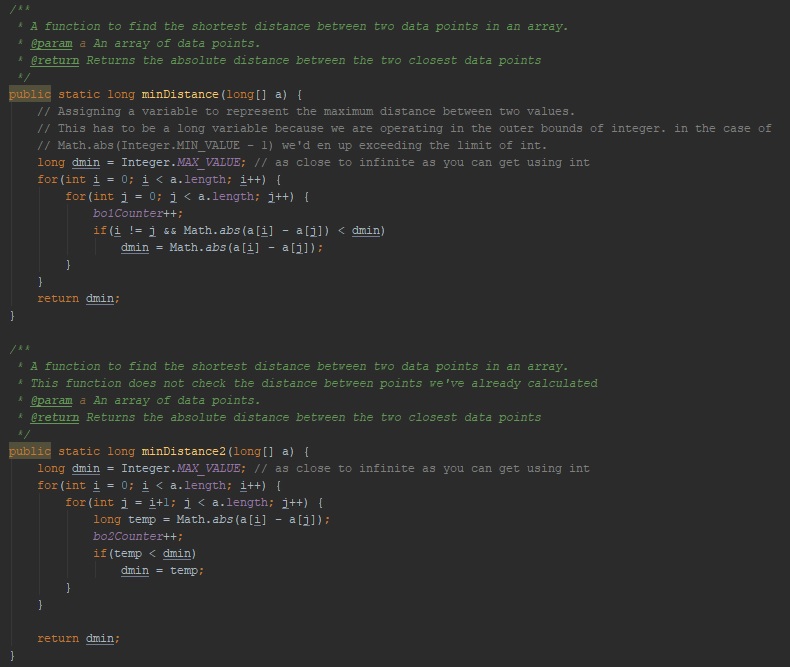
# Implementation of the algorithm

## Our code implementation

****

## 

## Proof of correctness

To test that our algorithms were correctly implemented we created our own test class. Using JUnit’s testing utility[[1]](#endnote-1) we created a test to check for possible errors in the algorithms. We tested that:

* The algorithms gave us the correct value of the shortest distance between values.
* Negative numbers were correctly handled.
* Equal values would give 0 as distance.
* Equal distances would still provide correct answers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case** | **Test instance** | **Expected result** | **Actual output** | **Test result** |
| Shortest distance between values found | A = [10, 4, 8, 1, 2] | 1 | 1 | Passed |
| Negative numbers handled correctly | A =  [0, 500, 1000, -501, 10] | 10 | 10 | Passed |
| Multiple occurrences of shortest distance | A = [0, -1, -2, -3, 1, 2, 3] | 1 | 1 | Passed |
| Equal values correctly handled | A = [10, 0, 3, 1, 0, 6, 0] | 0 | 0 | Passed |

## Methodology, tools and techniques

We chose Java as our coding language because it’s the language we’ve both had the most experience with and the language where we could represent the algorithm as accurate as we could. Java is an object oriented programing language provided by Oracle and is one of the most popular programming languages as of early 2018[[2]](#endnote-2).

The experiment was performed on one of QUT’s S-block computers. It runs 64x Windows 10 Enterprise with an Intel® Core™ i7-6700 CPU @ 3.40GHz processor*[Appendix #1]*. Java’s pseudorandom Random class was used to create the test data. For estimating time we used Java’s innate System class. It has a function to return the current system’s time in millisecond. In order to assure a minimum of interference from other processes we closed as many other processes as we could while the experiment was running.

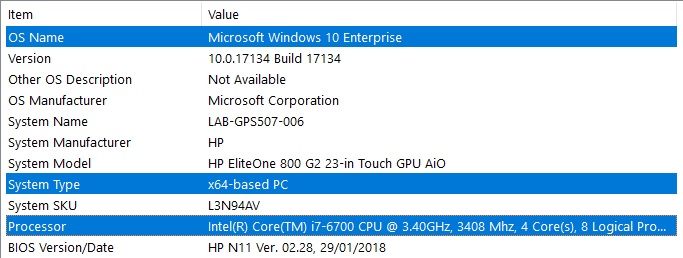
For producing the graphs we ran our java code that writes array sizes, basic operations counted and time spent for each run through of the algorithm to two separate files. One for Levitin’s algorithm and one for the second provided algorithm. Then we plotted our test results using Microsoft® Office Excel 2016.

## Producing test data

For producing the test data I wrote a function to create an array containing random integers given a certain range of numbers and array size. When calling this function I set the same value for size and maxVal. After finishing the experiment I considered making the maxVal a constant equal to the maximum value of int to assure maximum variety in the values because having the same number occur multiple times means we have a higher chance of finding the median earlier in the array, which can reduce computation times. I chose not to make this change due to time limitations and the fact that the assignment does not make any restrictions towards creating the test input. I am however aware that this can lead to reduced average computation time.

# Appendix

#1



1. http://junit.sourceforge.net/javadoc/org/junit/Assert.html [↑](#endnote-ref-1)
2. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ [↑](#endnote-ref-2)