# SENG3320 Assignment 2

## Task 1 – Fuzz Testing

### Blackbox Testing

The program runs on a range of input based on some possible partitions. These include files which are empty, contain many characters (100000), contain many lines (10000), all possible files which contain a single character, and duplicate input. Running the program will print which values are being tested, as well as any errors that are caught.

### Performance Testing

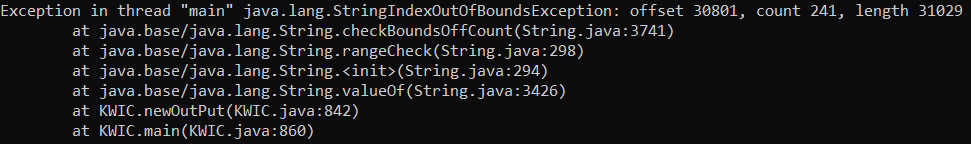
The table below shows the program running with the file input duplicated several times, and with incrementally smaller input (removing lines from the end of the file) to show the different execution duration totals after 10 runs.

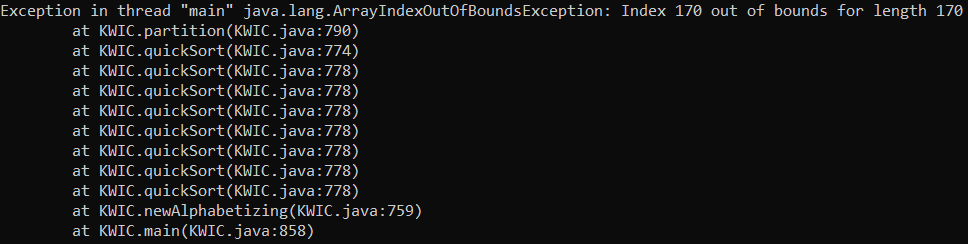
|  |  |
| --- | --- |
| Program Execution Duration on books.txt | |
| Number Of Lines | **Duration (seconds)** |
| 2220 | 0.7276 |
| 1110 | 0.6331 |
| 555 (original) | 0.2021 |
| 444 | 0.1894 |
| 333 | 0.1967 |
| 222 | 0.1550 |
| 111 | 0.1377 |

### Fuzz Testing

The fuzz testing tool randomly generates a file between 0 and 5000 characters long. The file can be made up of all characters found on a keyboard (0-9, a-z, A-Z, brackets, space etc) and the new line character. This ensures the number of lines in the file is also random. The file is then run on the KWIC program. If a new error is found, it adds it to the list of errors, saves the file that caused the error, and continues running. After randomly creating and testing 1000 files, the program halts.

### Errors Found





### Report

The initial program generated a random string of 100000 characters (including capitals, symbols, space, new line etc) and saved them to a txt file. This file was passed to the KWIC program and the errors were recorded. As new errors were found, the offending line was removed from the file and the test was run again.

This proved very difficult as without knowing how the program worked, it was difficult to locate the line that caused the error, and removing lines manually was very time consuming. To resolve this, new test cases were generated and run for every test. The generated string was changed to be 10000 characters long to improve computation time, and if an error occurred, the file that threw the error was saved and a new test file was generated and tested. This approach caught 2 of the errors, but ultimately was too slow.

It was then suggested to try using the java.lang.Runtime library to run the program from within another program. This was very useful for performance testing as it allowed for tests to be run and recorded automatically. The program was changed to generate a file with a random string of characters of length between 0 and 5000. If a new error is found, it saves the file and continues looping through the programs. After over 10000 tests, the only errors that were found we the errors that had already been found during the manual random testing.