1. **Introduction**
   This is the report detailing how I parallelised a Least Significant Digit radix sort.

2. **User guide**
   Compile -> javac *.java
   Parameter line arguments -> java Oblig4 [n] [seed] [number of threads]
   Example -> if you wanted n = 1000, seed = 28 and threads = 4 the command would be “java Oblig4 1000 28 4”

3. **Parallel Radix sort – how parallelisation of part C was done**
   First, each thread fills in n/k of the sumCount[] array by summing up the relevant parts of the allCount[][] array. Once the sumCount[] array has been filled in by all the threads, each thread fills in its localCount[] array by taking the value at the corresponding index from localCount[] and adding to it the values from allCount[][] from all of the threads that have a lower index than the current thread.

4. **Implementation – how the programme works and was tested**
   The programme works. At first, I tried to write separate Thread classes for each stage of the parallelisation, but I quickly realised that having all of the functionality in one Thread class would work better. Other than that, writing this programme was fairly straightforward. I didn’t really do any more testing than printing parts of the programme to the terminal.

5. **Measurements – specs, tables, graphs of speedup, discussion**
   Specs: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 2801 Mhz, 4 cores.

   Table and graphs on last page of report.

   As can be seen from the table and graph the parallel code is much slower than the sequential code when N < 1,000,000. Once at 10,000,000 and 100,000,000 it stays easily around 2-3x the speed. I imagine this is because the time spent creating the threads and synchronising is more overhead than it’s worth when sorting smaller arrays.

6. **Conclusion**
   I have managed to write code that uses multiple threads to sort arrays with sizes up to 100,000,000. This code does it faster than the sequential implementation. In the process I have learnt to think in a better way about how and where to synchronise.

7. **Appendix**
   N = 1000
   Sequential: 0,11ms Parallel: 2,54ms Speedup: 0,0
   Sequential: 0,08ms Parallel: 2,51ms Speedup: 0,0
   Sequential: 0,09ms Parallel: 1,83ms Speedup: 0,0
   Sequential: 0,09ms Parallel: 1,34ms Speedup: 0,1
   Sequential: 0,09ms Parallel: 1,06ms Speedup: 0,1
   Sequential: 0,09ms Parallel: 1,02ms Speedup: 0,1
   Sequential: 0,09ms Parallel: 1,01ms Speedup: 0,1
   Median -> Sequential: 0,09ms Parallel: 1,34ms Speedup: 0,1

   N = 10,000
   Sequential: 0,90ms Parallel: 3,56ms Speedup: 0,3
   Sequential: 0,93ms Parallel: 3,32ms Speedup: 0,3
   Sequential: 0,29ms Parallel: 2,64ms Speedup: 0,1
   Sequential: 0,29ms Parallel: 2,18ms Speedup: 0,1
   Sequential: 0,25ms Parallel: 1,78ms Speedup: 0,1
   Sequential: 0,23ms Parallel: 2,04ms Speedup: 0,1
   Sequential: 0,33ms Parallel: 2,98ms Speedup: 0,1
   Median -> Sequential: 0,29ms Parallel: 2,64ms Speedup: 0,1

   N = 100,000
   Sequential: 8,08ms Parallel: 11,64ms Speedup: 0,7

Sequential: 6,34ms Parallel: 12,29ms Speedup: 0,5
Sequential: 0,94ms Parallel: 7,82ms Speedup: 0,1
Sequential: 0,73ms Parallel: 3,45ms Speedup: 0,2
Sequential: 0,57ms Parallel: 3,71ms Speedup: 0,2
Sequential: 0,59ms Parallel: 2,63ms Speedup: 0,2
Sequential: 0,58ms Parallel: 1,80ms Speedup: 0,3
Median -> Sequential: 0,73ms Parallel: 3,71ms Speedup: 0,2

N = 1,000,000
Sequential: 24,32ms Parallel: 58,49ms Speedup: 0,4
Sequential: 21,74ms Parallel: 7,81ms Speedup: 2,8
Sequential: 7,15ms Parallel: 7,90ms Speedup: 0,9
Sequential: 7,23ms Parallel: 6,82ms Speedup: 1,1
Sequential: 7,60ms Parallel: 7,40ms Speedup: 1,0
Sequential: 6,76ms Parallel: 6,03ms Speedup: 1,1
Sequential: 12,06ms Parallel: 5,80ms Speedup: 2,1
Median -> Sequential: 7,60ms Parallel: 7,40ms Speedup: 1,0

N = 10,000,000
Sequential: 113,38ms Parallel: 90,05ms Speedup: 1,3
Sequential: 104,31ms Parallel: 55,91ms Speedup: 1,9
Sequential: 95,80ms Parallel: 41,35ms Speedup: 2,3
Sequential: 98,94ms Parallel: 42,99ms Speedup: 2,3
Sequential: 99,13ms Parallel: 44,80ms Speedup: 2,2
Sequential: 100,84ms Parallel: 38,16ms Speedup: 2,6
Sequential: 101,68ms Parallel: 38,42ms Speedup: 2,6
Median -> Sequential: 100,84ms Parallel: 42,99ms Speedup: 2,3

N = 100,000,000
Sequential: 1133,11ms Parallel: 449,84ms Speedup: 2,5
Sequential: 1348,18ms Parallel: 403,88ms Speedup: 3,3
Sequential: 1371,73ms Parallel: 376,73ms Speedup: 3,6
Sequential: 1237,46ms Parallel: 375,44ms Speedup: 3,3
Sequential: 1220,92ms Parallel: 336,65ms Speedup: 3,6
Sequential: 1161,42ms Parallel: 348,24ms Speedup: 3,3
Sequential: 1125,67ms Parallel: 391,86ms Speedup: 2,9
Median -> Sequential: 1220,92ms Parallel: 376,73ms Speedup: 3,2

N	Sekvensiell (ms)	Parallel (ms)	Speedup
1000	0,09	1,34	0,1
10000	0,29	2,64	0,1
100000	0,73	3,71	0,2
1000000	7,60	7,40	1,0
10000000	100,84	42,99	2,3
100000000	1220,92	376,73	3,2