Darren Redmond – 92026265 – Data Programming with C Exam

Q2. Benchmark the three sorting functions(quicksort, SortRcpp, SortSTL) against each other, sorting the vector x, and discuss the results

Well when I ran the benchmark code with 10000 elements in my sample the benchmark code took quite a long time to run.

```
benchmark(quickSort(x), SortRcpp(x), SortSTL(x), order = "relative")
```

Sort either one, two, or three of the algorithms is very inefficient. I am going to guess from previous benchmarks that the R code on 10000 elements providing a quick sort algorithm if probably not that quick compared to the others.

```
          test replications elapsed  relative user.self sys.self user.child sys.child
3   SortSTL(x)          100   0.009     1.000     0.008    0.001          0         0
2  SortRcpp(x)          100   0.222    24.667     0.213    0.004          0         0
1 quickSort(x)          100 111.079 12342.111    94.864   15.690          0         0
```

So after 111 seconds the R implemented quicksort algorithm returned. It is over 12000 times slower than using the sort with stl library, and 500 times slower than my Rcpp algorithm.

The STL code was almost 25 times faster than my implementation for SortRCpp but I haven't had a chance to tune it yet to make it faster. I am sure in the 20 minutes or so to get this code working – I didn't actually step back to see if I could tune it – but I would either do that – or just use the stl sort function – as even if I improved my code significantly SortSTL is still going to be sufficient to do the sort – doing it in nine – one thousandths of a second to sort a 10000 element vector.

So I would recommend going with the stl approach to calculate this a lot less code, and a lot quicker.

Q1 A) – With only 1000 steps and 100000 attempts, not once did my simulation find the correct solution. So the approximate probability of finding the correct location was 0.

Q1 B) I created a grid to calculate the walk of 1000 x 1000 squares and centred the starting point at 500, 500, so there was no possibility of walking off the grid and causing a seg fault in the code. I randomly select a direction with the rand function and then would update the curX and curY variables and after each step check them for the desired location and exit true if found location otherwise take another step. If a location was found then the algorithm would return true, and also kept track of each grid location – in case I would want to plot that later – however this version of the code does not do that.