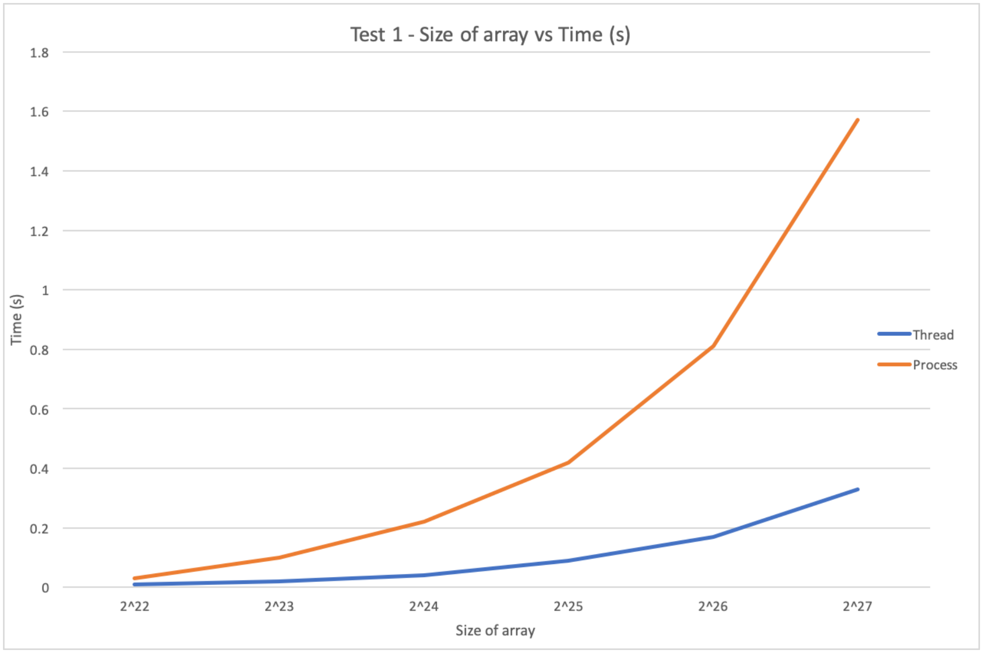
Test 1

Test 1 looks at how varying the size of the list effects runtime when there is a fixed number of processes/threads. Lists of size 2^22, 2^23, ... 2^27 were used, each being run by 16 total processes/threads.

The results were that both threads and functions slowed down as a result of the increase in size. This confirmed out expectations of how threads and processes would behave as the size of the array they iterate over increases.

One thing to note however, as the lists sizes doubled, so did the average runtime for Threads. Processes however, grew much more rapidly as shown in the graph below.

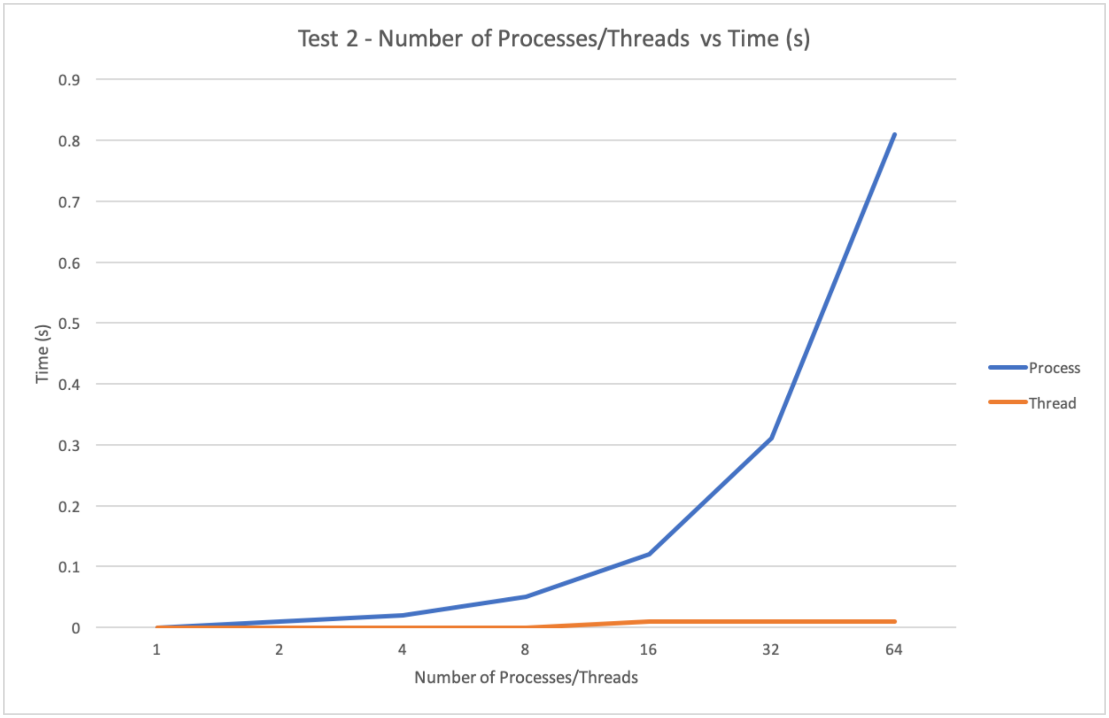


Test 2

Test 2 looks at how increasing the number of processes/threads over a list of a fixed length effects runtime. We chose a relatively large list for this test, using a list of size 2^16. For the number of processes/threads we chose sizes 1, 2, 4, 8, 16, 32, and 64.

Despite the number of threads/processes, the original runtime for this test gave very small time values, where the maximum value returned was .01 seconds. So to make the data more meaningful, we measured how long it would take to search through the list 10 times.

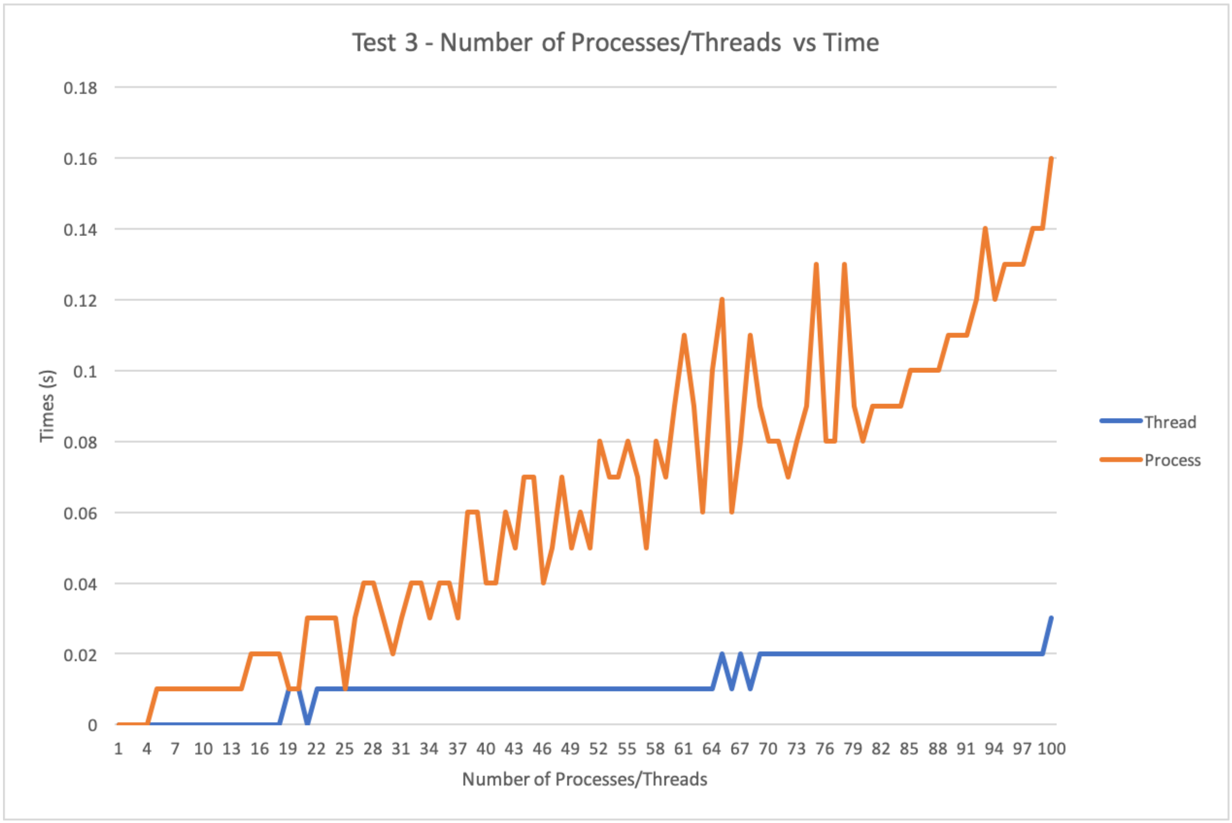
Threads were pretty consistent, not slowing down much at all as the number of threads increased. Processes however, took much longer to return as the number of processes increased. Though it was surprising that threads did not grow much, the increase in time for processes we found was expected.



Test 3

Because there was such a small increase in time as the number of threads increased in Test 2, we wanted to investigate what would happen if the number of threads approached the size of the list. We thought perhaps if the ratio of threads to items is the list was large, that we would start to see the time efficiency of threading slow down. So in test 3, we iterated the number of processes/threads from 1 to 100 and searched a list of size 100.

Though only a small increase in time was found for threads, the average time returned by threads did increase slightly as the number of threads increased. Processes however grew at a much faster rate.



Challenges

One issue we had was measuring a consistent runtime for processes. In the test data shown above in each of the three test cases, we had to retrieve this data by isolating the test functions into separate programs. We noticed the longer the program ran, and the more processes it created, the longer it would take to create a new process. This meant if the program ran Test 1 and Test 2 first, Test 3 would spend much more time than it would if it were running on its own.

The program can run all three test cases at once, however by the time the program reaches Test 3 it takes a very long time to print the data.

This may explain why in Test 1 processes grow so much more rapidly as the size doubles, despite the number of processes remaining the same.