1. a. ./test1 16 20 8

```
      Total time (s) for j=1 is 1.161467 with 268435456 accesses
      Total time (s) for j=2 is 1.306814 with 268435456 accesses
      Total time (s) for j=4 is 1.623957 with 268435456 accesses
      Total time (s) for j=8 is 1.539909 with 268435456 accesses
      Total time (s) for j=16 is 1.262742 with 268435456
accesses
      Total time (s) for j=32 is 1.074793 with 268435456
accesses
      Total time (s) for j=64 is 0.570838 with 268435456
accesses
      Total time (s) for j=128 is 0.936864 with 268435456
accesses
```

   b. ./test2 16 28 8

```
      Total time (s) for j=1 is 2.935848 with 268435456 accesses
      Total time (s) for j=2 is 1.408126 with 268435456 accesses
      Total time (s) for j=4 is 0.943753 with 268435456 accesses
      Total time (s) for j=8 is 1.026748 with 268435456 accesses
      Total time (s) for j=16 is 0.813783 with 268435456
      accesses
      Total time (s) for j=32 is 0.788916 with 268435456
      accesses
      Total time (s) for j=64 is 0.798105 with 268435456
      accesses
      Total time (s) for j=128 is 0.799474 with 268435456
      accesses
```

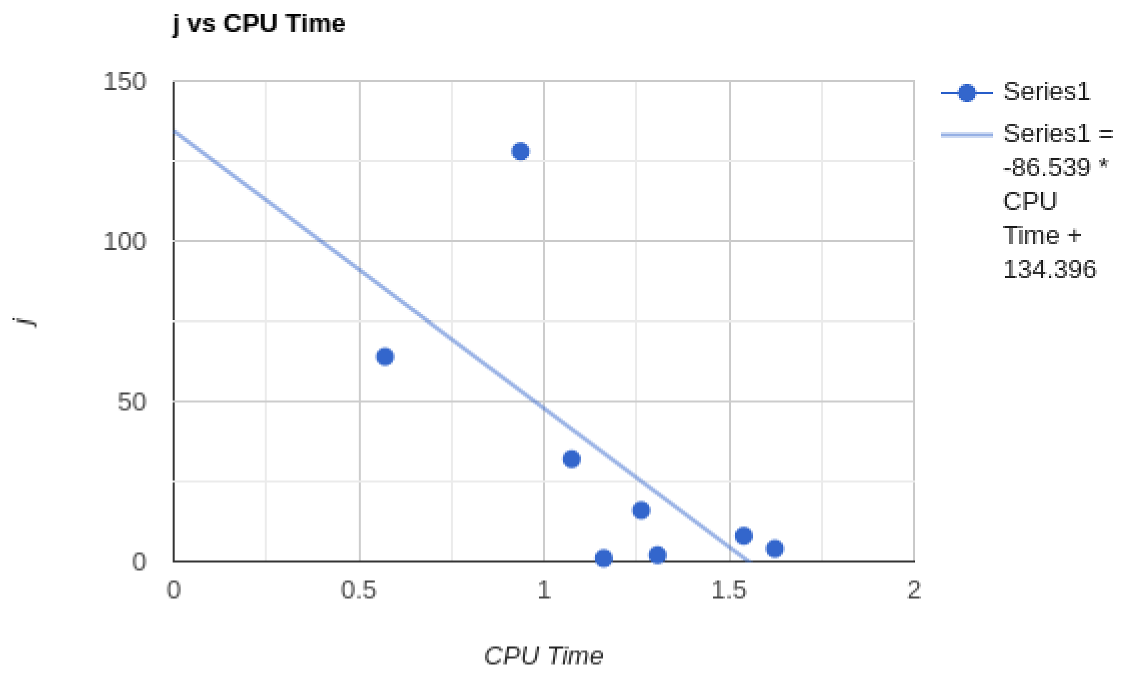
   c. ./test3 16 20 8 2

```
      Total time (s) for j=1 is 2.941135 with 528482304 accesses
      Total time (s) for j=2 is 2.891837 with 528482304 accesses
      Total time (s) for j=4 is 2.889403 with 528482304 accesses
      Total time (s) for j=8 is 2.202141 with 528482304 accesses
      Total time (s) for j=16 is 1.772445 with 528482304
      accesses
      Total time (s) for j=32 is 1.809743 with 528482304
      accesses
      Total time (s) for j=64 is 1.818033 with 528482304
      accesses
      Total time (s) for j=128 is 1.822416 with 528482304
      accesses
```

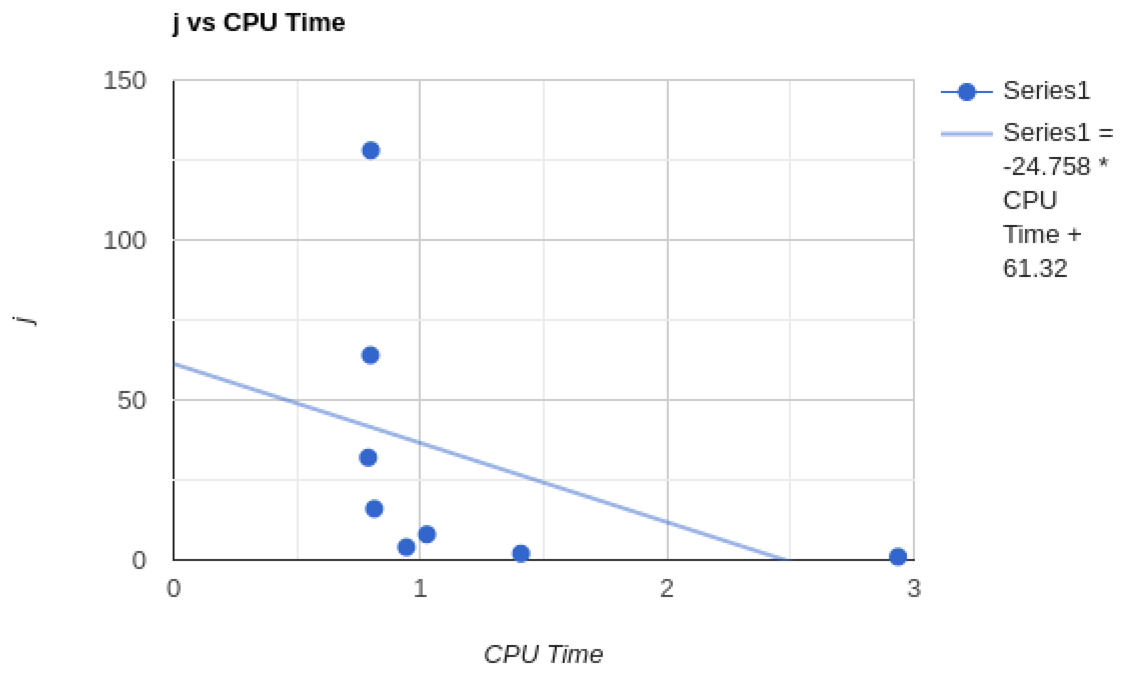
2. For the number of experiments done… the less amount you use in an array the faster the experiment will perform with less accesses. In the for loop when changing the value of N (for

test values) for the array it raises and lowers the amount of time for each test.

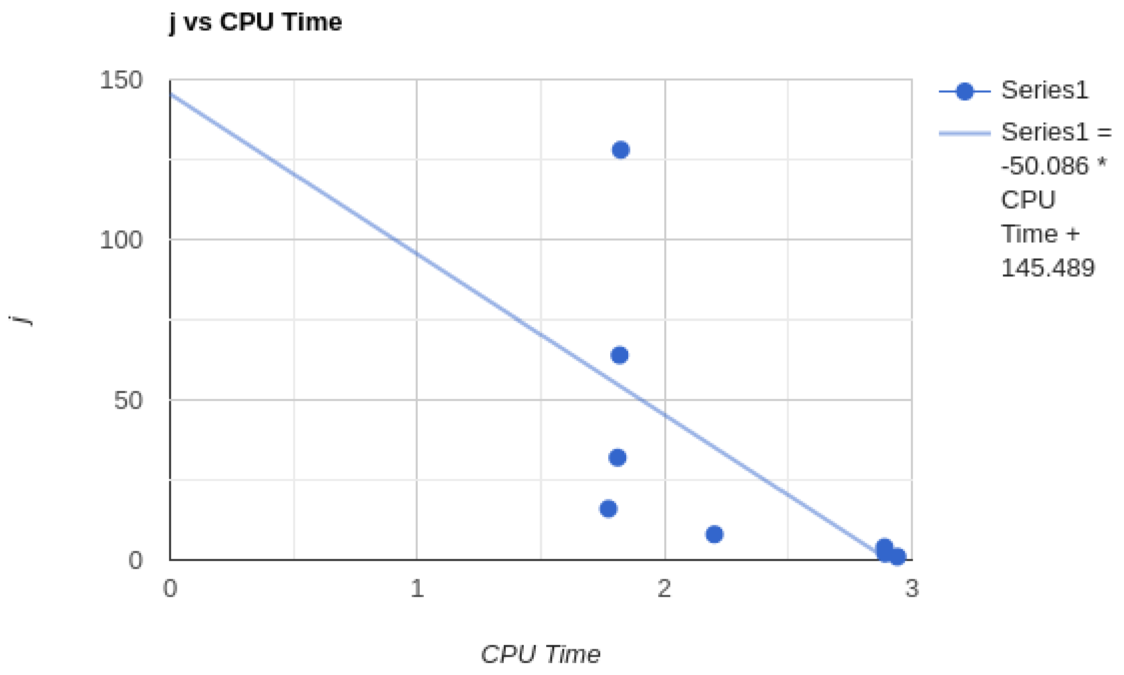
3. I used the same number given in the example, and I know it has L1 cache because access times are below 2 seconds and roughly around 1 second. This is demonstrated by having a point significantly below 1 second and lots of points being hit regularly around 1 second.



4. Same variables as example, but its slower due to the change in variables and uses the same L1 cache; this is demonstrated by the consistent data points below 1 value(s). L2 and L3 cache was also demonstrated but i didn’t put the graphs here



5. As the amount of data increases and the number increases the time tends to increase even on L1 cache. There is a subtle outlier here but the one closest to the line and the consistent times j to CPU time make it fast but consistent cache. This could lead to some sort of associative cache. There were other test that showed slower times including those from L2 and L3 cache but those weren’t made to make things simpler.



6. I think this cache might be 8-way associative, maybe 16-32 KB of memory for the cache.
    a. Actual stats 8x32KB, 8-way associative, 64-bytes line size for L1
    b. L2 = 8x512 KB 8-way associative, 64-bytes line size
    c. L3 = 8 MB, 16-way associative, 64-byte line size
    d. The Processor I’m running is a Ryzen 9 4900HS 16 cores
        i. Yes this does end up lining up with my initial experimentations, even though i might have been slightly unsure about the cache memory size