**Experimental Analysis**

**ALGORITHM 1**

**1)**

|  |  |
| --- | --- |
| Enumerative | |
| N | Time |
| 20 | 0.00008 |
| 50 | 0.000587 |
| 100 | 0.003105 |
| 200 | 0.018828 |
| 300 | 0.052911 |
| 500 | 0.215364 |
| 1000 | 1.461163 |
| 1200 | 2.596658 |
| 1500 | 4.956266 |
| 2000 | 11.66336 |

**2)**

3)

The curve is quadratic with an exponent value of 3, which is O(n^3).

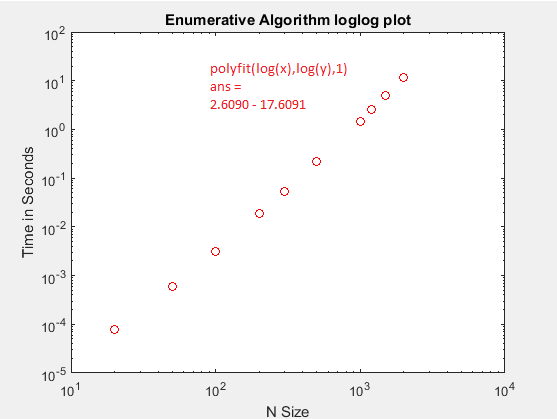
4)

Since the slope of our loglog plot below is 2.6090, then this gives us a run time of n^2.6090, but this is still O(n^3) since big O notation assumes that n^3 is the worst case scenario. This is true in this case since the 2nd and 3rd loops are not fully traversed each time.

5)

10 minutes = 600 seconds, so I took 600 seconds and set it equal to 1E-09x3 + 4E-08x2 + 3E-05x + 5E-05, and plugged this into Wolfram Alpha. The result was x = 8419.8. So the greatest size of N that could be solved using this algorithm in 10 minutes would be roughly 8419.6.

6)



Slope = 2.6090, which would mean our algorithm is O(n^2.6090) according to the loglog plot.

**ALGORITHM 2**

**1)**

|  |  |
| --- | --- |
| Iterative | |
| N | Time |
| 20 | 0.0000256 |
| 50 | 0.0001441 |
| 100 | 0.0003382 |
| 200 | 0.0012358 |
| 300 | 0.0024545 |
| 500 | 0.0081496 |
| 1000 | 0.0295787 |
| 1200 | 0.0410674 |
| 1500 | 0.0622348 |
| 2000 | 0.1161804 |

**2)**

3)

The curve is quadratic, which is O(n^2)

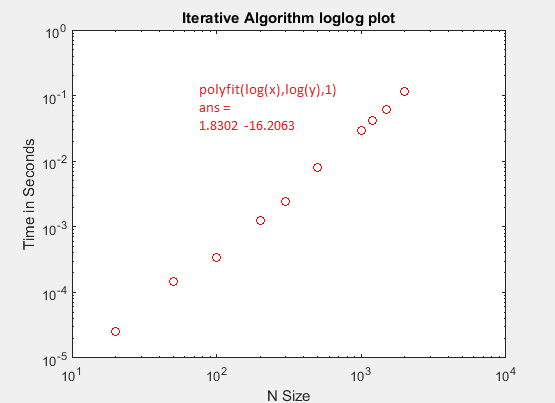
4)

The slope of the loglog plot was O(n^1.8302), so our experimental runtime was the same as our theoretical, which was O(n^2). No discrepancies.

**5)**

Using Wolfram Alpha, I set 600 = 3E-08x2 - 1E-06x + 0.0003. The result was x = 141,438. So the greatest value of N this algorithm could solve in 10 minutes is roughly 141,438.

**6)**

slope = 1.8302

**ALGORITHM 3**

**1)**

|  |  |
| --- | --- |
| Divide & Conquer | |
| N | Time |
| 20 | 0.00006 |
| 50 | 0.000169 |
| 100 | 0.000487 |
| 200 | 0.001543 |
| 300 | 0.003079 |
| 500 | 0.007769 |
| 1000 | 0.030524 |
| 1200 | 0.043727 |
| 1500 | 0.070405 |
| 2000 | 0.127336 |

**2)**

3)

The curve isn’t quite linear, but it is closest to being linear.

4)

The slope of our loglog plot was m ~ 1.7041, we can solve for an equation of the form T(n) = cn log n where c is some constant:

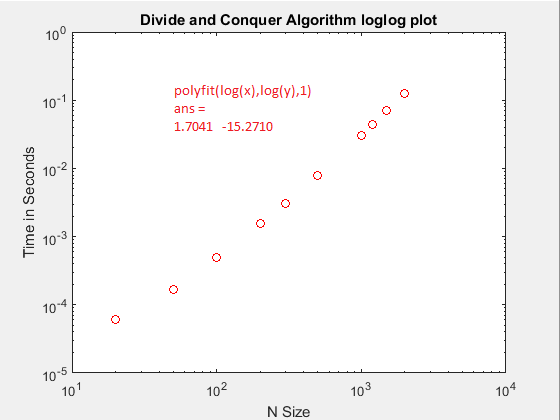
T(n) ~ .0000017041 \* n \* log n

Any discrepancies could be slight variations of runtime, could also depend on your system speed.

**5)**

Using Wolfram Alpha, we plugged in 600 = 6E-05x - 0.0113 and got x = 10,000,200. So in 10 minutes, the max value of N this algorithm could solve is around 10,000,200.

**6)**

Since the slope is 1.7041, the algorithm runs in .0000017041 \* n log(n)

**ALGORITHM 4**

**1)**

|  |  |
| --- | --- |
| Linear | |
| N | Time |
| 20 | 0.000003 |
| 50 | 0.000005 |
| 100 | 0.00001 |
| 200 | 0.00001 |
| 300 | 0.00002 |
| 500 | 0.00003 |
| 1000 | 0.00007 |
| 1200 | 0.00008 |
| 1500 | 0.000116 |
| 2000 | 0.000169 |

**2)**

3)

Find a function that models the relationship between input size n and time. This function will produce a curve that “fits” the data you plotted in part 2. To determine the equation of the function for the curve use regression techniques. The shape of the curve will determine the type of regression you use. Is the data linear/quadratic/logarithmic/exponential?

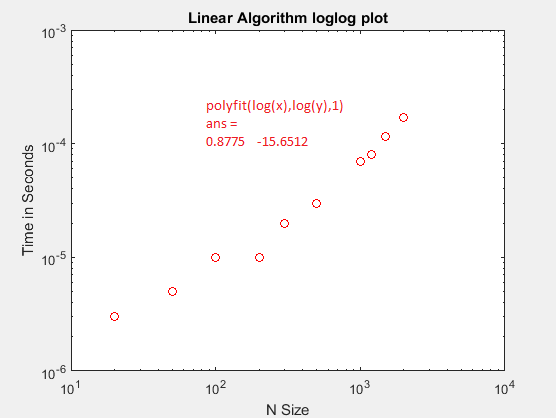
4)

Discuss any discrepancies between the experimental and theoretical running times

**5)**

Using Wolfram Alpha, we plugged in 600 = 8E-08x - 4E-06 and got x = 7,500,000,050. So in 10 minutes, the max value of N this algorithm could solve is around 7,500,000,050.

**6)**

Slope = .8775, so our algorithm runs in O(n^.8775) according to this loglog plot.

**7)**

**Plot of all four algorithms together**

