Yair Caplan

Intro to Algorithms

Order-of-Growth Analysis for Secret Algorithms

Description of Experimental Process

I first warmed up the system by running Cinebench (R20). Next, I ran my Java code, where I followed the following procedure for each secret algorithm:

- Execute the algorithm for n = powers of 2, starting with values where the time is non-zero, up until the values where it took a significant amount of time to execute (~10^5 ms/execution)
- Repeat the execution 3 times for each value of n, and use the average time for the data analysis, in order to smooth out the data a bit
- Print a row for each value of n with the time in milliseconds and the ratio to the previous n

Finally, using the resulting table, I plotted $\log_2$(time) against $\log_2$(n), and used an online tool[1] to fit a line to it using linear regression.

NOTE - - in the graphs, the axes labeled as 'Y-DEPENDENT' are $\log_2$(time) (ms), and the axes labeled 'X-INDEPENDENT' are $\log_2$(n).

Contents:

[1] https://www.socscistatistics.com/tests/regression/default.aspx

Secret Algorithm 1

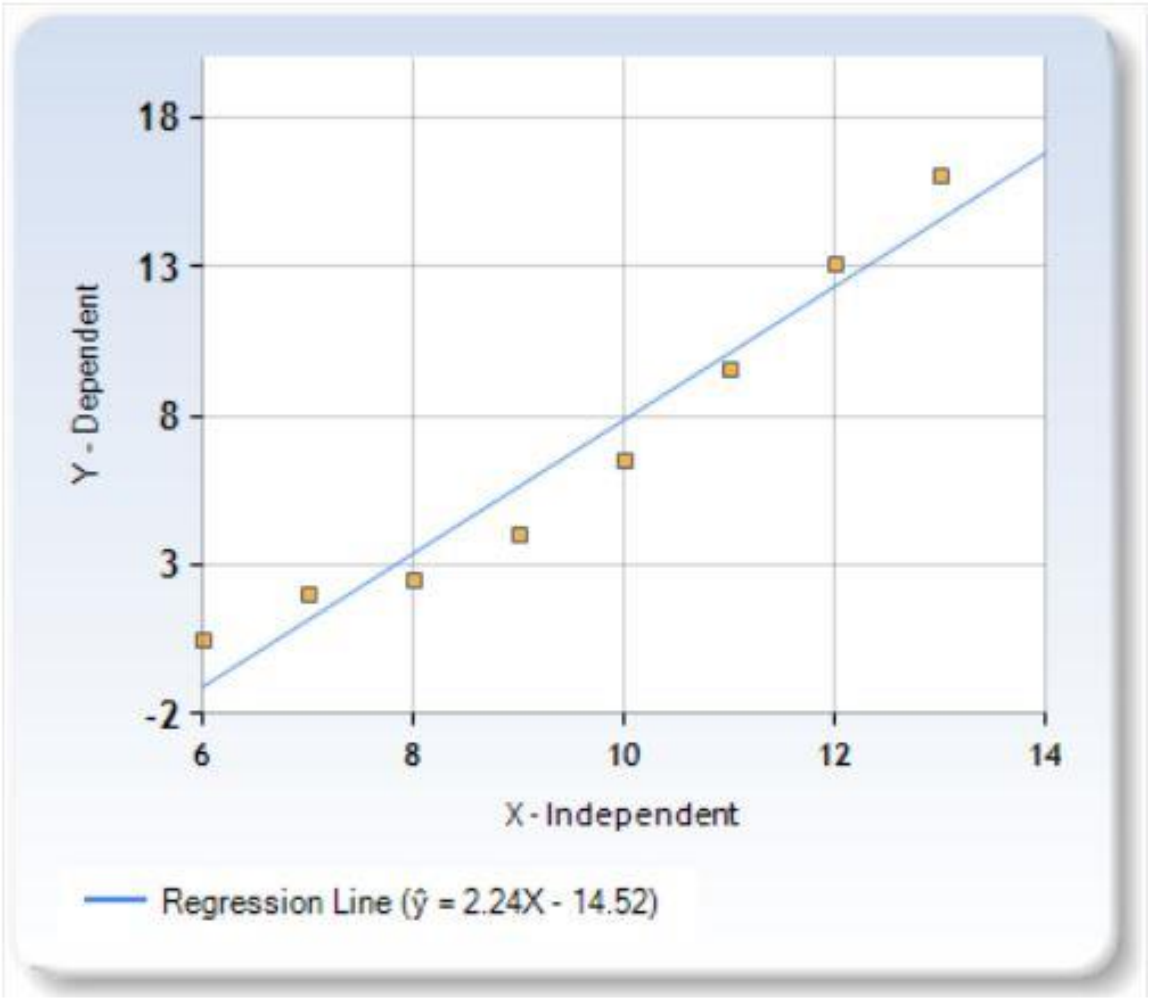
Performance Analysis Conclusion => *O(n³)*

Tables:

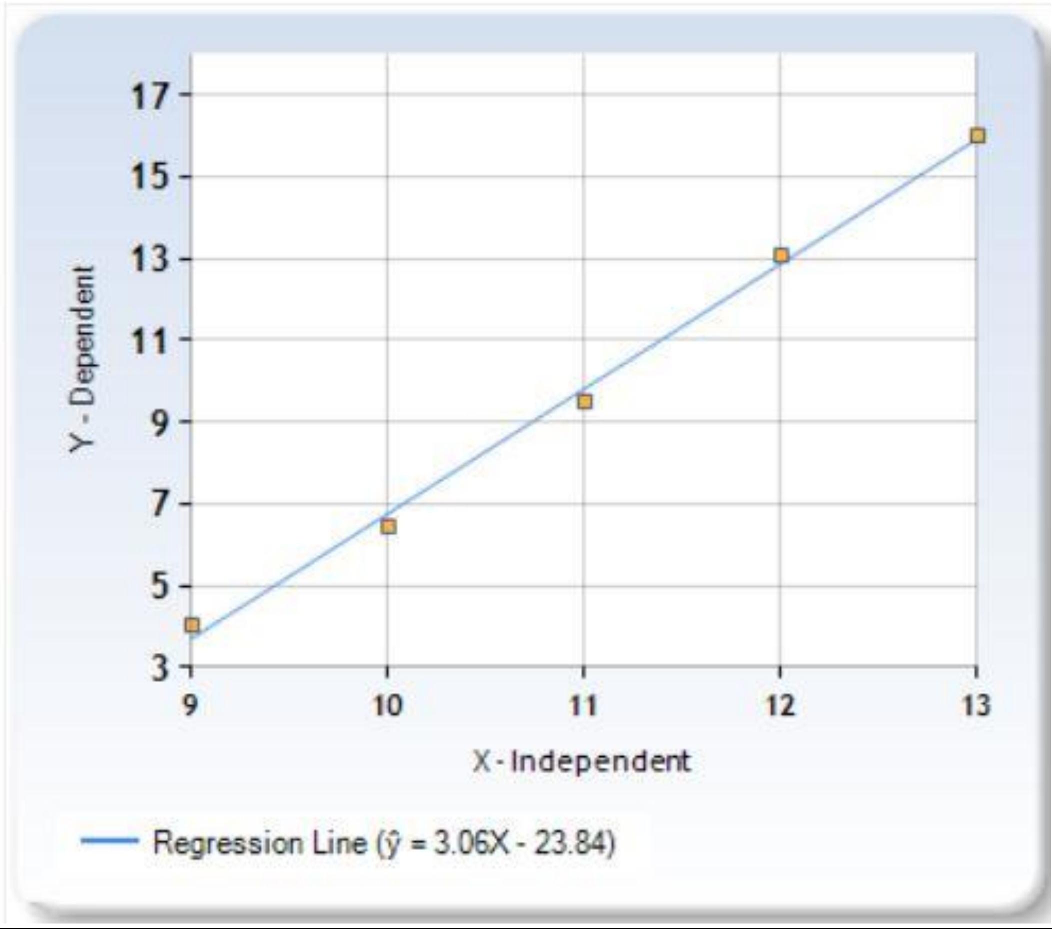
Code Output:

```
**********
SecretAlgorithm1 Results
**********
32 (n)   0.333333 time (ms)   Infinity (ratio)
64 (n)   1.333333 time (ms)   4.000000 (ratio)
128 (n)   4.000000 time (ms)   3.000000 (ratio)
256 (n)   5.666667 time (ms)   1.416667 (ratio)
512 (n)   16.333333 time (ms)   2.882353 (ratio)
1024 (n)   88.333333 time (ms)   5.408163 (ratio)
2048 (n)   731.333333 time (ms)   8.279245 (ratio)
4096 (n)   8469.000000 time (ms)   11.580219 (ratio)
8192 (n)   67280.666667 time (ms)   7.944346 (ratio)
```

log-log graph using all the data points except the smallest:



log-log graph using only the largest 5 data points:



Reasoning:

The results are (ostensibly) not as nice and neat as the other experiments, let alone the textbook. However, it would seem that the ratios hover around 8. Ignoring the cases that took less than 10 milliseconds to execute, the average ratio is ~7.25, which is close enough to 2^3 to safely say that this algorithms performs $O(n^3)$. The log-log graphs support this conclusion, since the slope with almost all the data points is over 2 (2.24), but when we focus in on the 5 largest data points the slope is almost exactly 3 (3.06).

Secret Algorithm 2

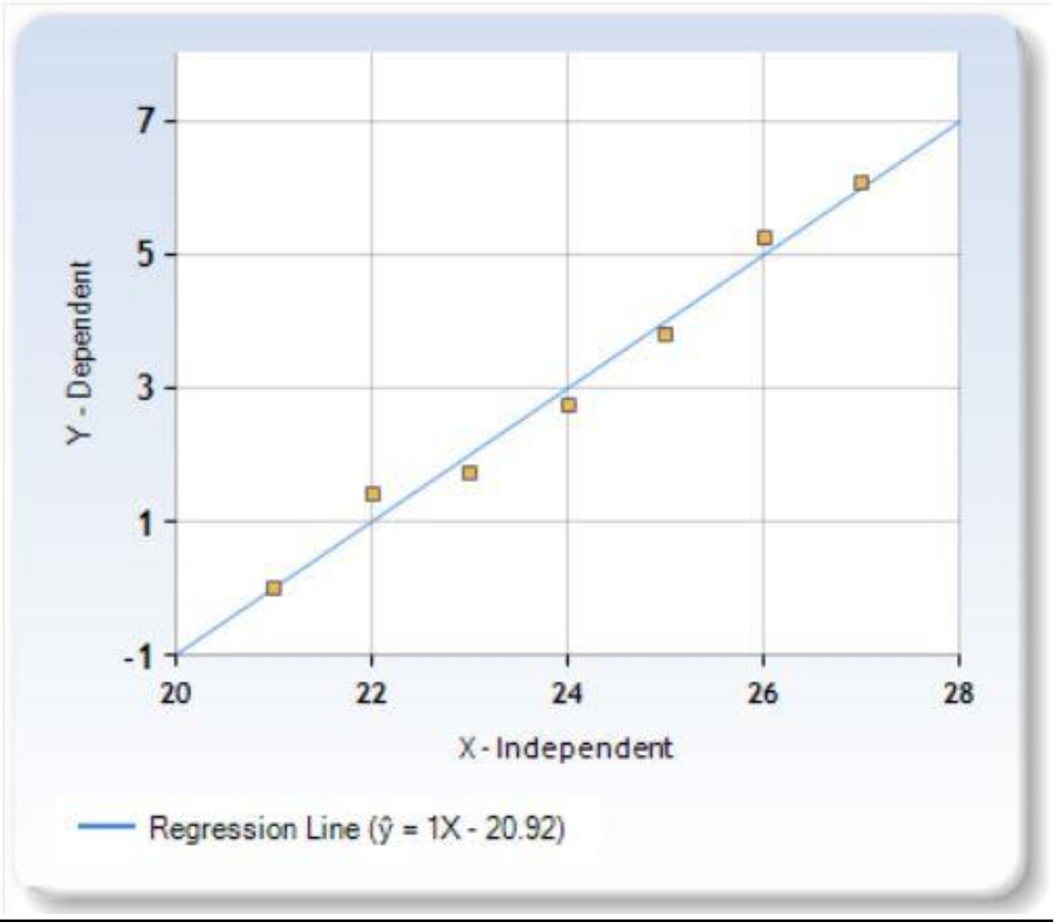
Performance Analysis Conclusion => O(n)

Tables:

Code Output:

```
**********
SecretAlgorithm2 Results
**********
131072 (n)   1.000000 time (ms)   Infinity (ratio)
262144 (n)   0.000000 time (ms)   0.000000 (ratio)
524288 (n)   0.000000 time (ms)   NaN (ratio)
1048576 (n)   0.666667 time (ms)   Infinity (ratio)
2097152 (n)   1.000000 time (ms)   1.500000 (ratio)
4194304 (n)   2.666667 time (ms)   2.666667 (ratio)
8388608 (n)   3.333333 time (ms)   1.250000 (ratio)
16777216 (n)   6.666667 time (ms)   2.000000 (ratio)
33554432 (n)   14.000000 time (ms)   2.100000 (ratio)
67108864 (n)   37.666667 time (ms)   2.690476 (ratio)
134217728 (n)   67.000000 time (ms)   1.778761 (ratio)
```

log-log graph using all the largest 7 data points:



Reasoning:

This algorithm ran very quickly on small data sets, and I had to increase n by a lot more (relative to algorithm 1) to get meaningful, non-zero times. But once it reached these times, the ratios quicky settled close to 2, or 2^1, indicating linear performance. However, the program would run out of heap-memory when setup() was called with n=2^{27}, even though the time was generally still under 70 ms and I would have ideally wanted to test until the cases began taking significant amounts of time (as mentioned in the 'Description of Experimental Process'). That said, the log-log graph strongly indicates linear performance, as the slope is 1. So I think my best guess is O(n).

Secret Algorithm 3

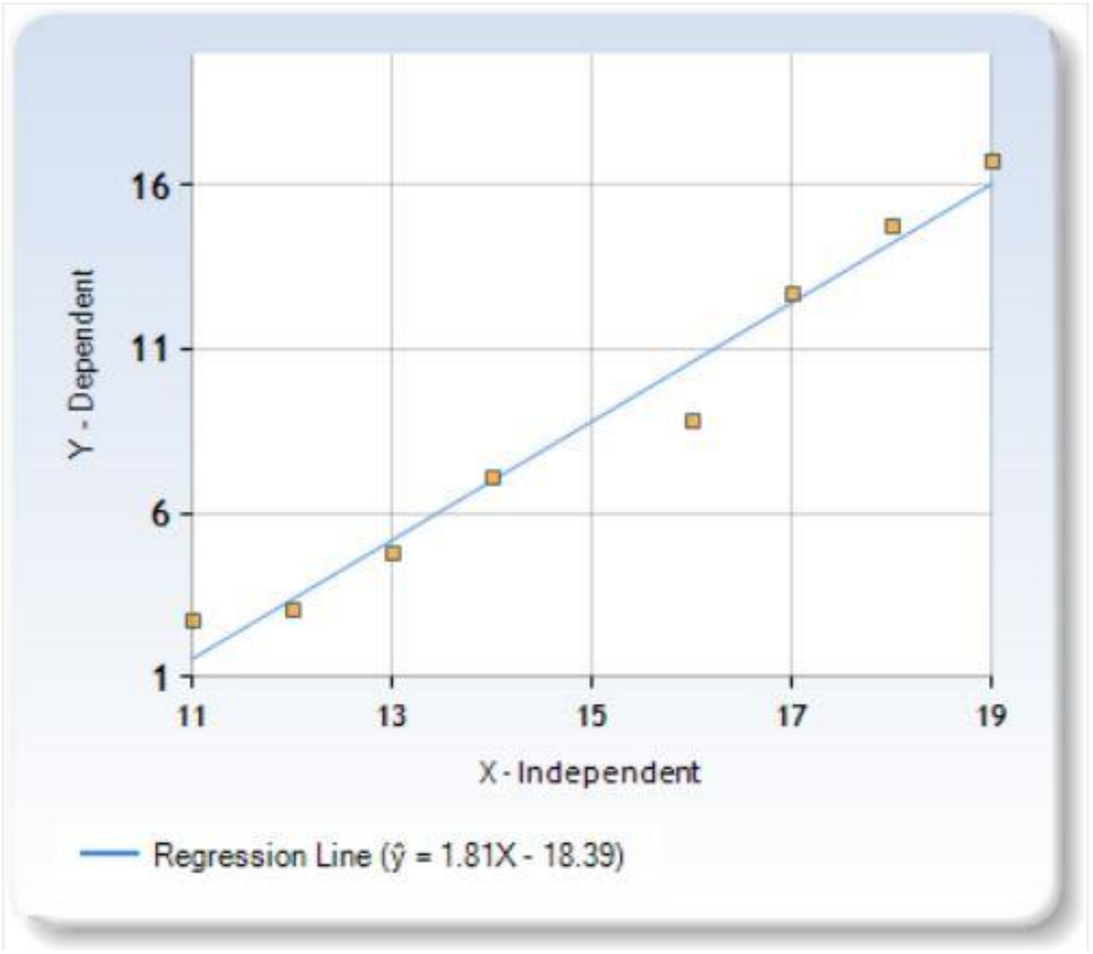
Performance Analysis Conclusion => *O(n^2)*

Tables:

Code Output:

```
**********
SecretAlgorithm3 Results
**********
2048 (n)   6.666667 time (ms)   Infinity (ratio)
4096 (n)   8.333333 time (ms)   1.250000 (ratio)
8192 (n)   26.666667 time (ms)   3.200000 (ratio)
16384 (n)   132.000000 time (ms)   4.950000 (ratio)
32768 (n)   440.000000 time (ms)   3.333333 (ratio)
65536 (n)   2885.333333 time (ms)   6.557576 (ratio)
131072 (n)   6467.000000 time (ms)   2.241335 (ratio)
262144 (n)   27713.333333 time (ms)   4.285346 (ratio)
524288 (n)   111870.666667 time (ms)   4.036709 (ratio)
```

log-log graph using all the data points, excluding the smallest case and one outlier (n=2^{15}):



Reasoning:

These were some of the cleanest results I had throughout this experiment with respect to the results table, as by the end the ratios settled at almost exactly 4, or 2^2. And although the log-log graph doesn't show exactly a slope of 2, I think 1.81 is close enough to conclude that this algorithm is quadratic.

Secret Algorithm 4

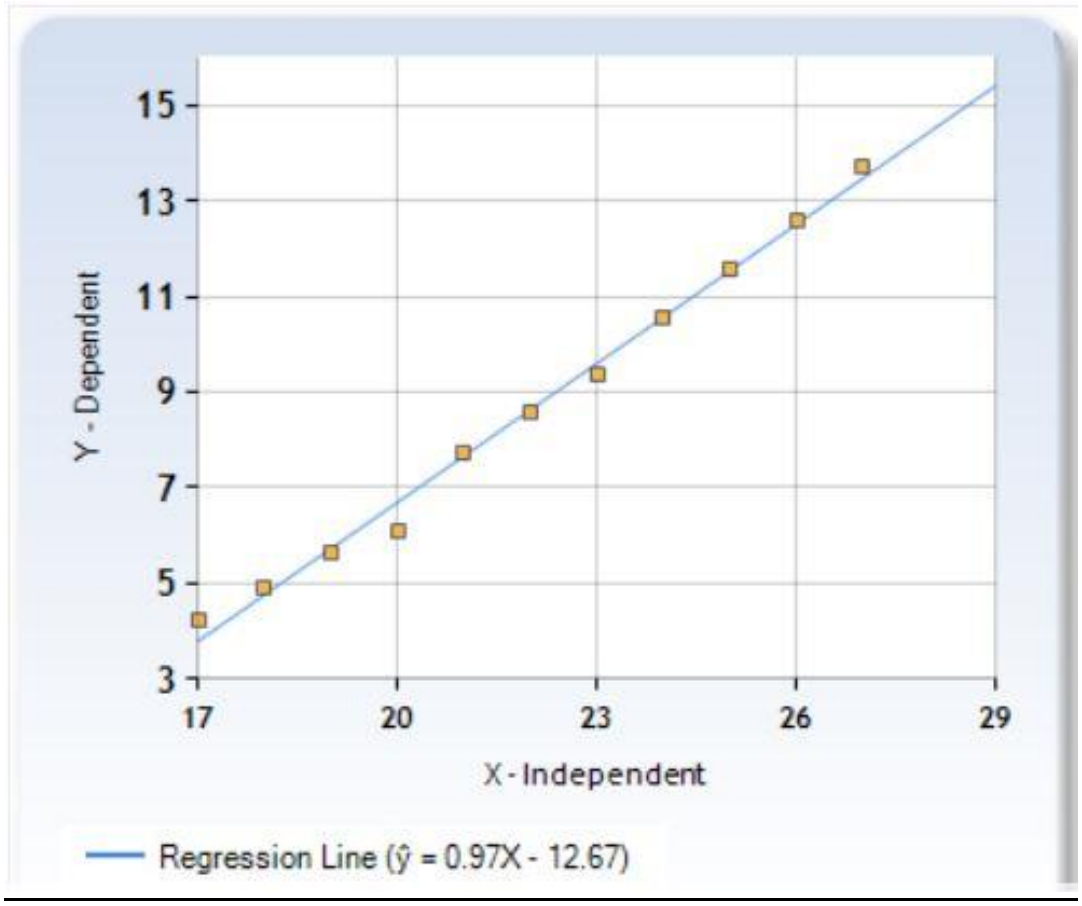
Performance Analysis Conclusion => *O(n)*

Tables:

Code Output:

```
**********
SecretAlgorithm4 Results
**********
131072 (n)   19.000000 time (ms)   Infinity (ratio)
262144 (n)   30.000000 time (ms)   1.578947 (ratio)
524288 (n)   50.000000 time (ms)   1.666667 (ratio)
1048576 (n)   67.666667 time (ms)   1.353333 (ratio)
2097152 (n)   209.333333 time (ms)   3.093596 (ratio)
4194304 (n)   387.333333 time (ms)   1.850318 (ratio)
8388608 (n)   658.333333 time (ms)   1.699656 (ratio)
16777216 (n)   1519.000000 time (ms)   2.307342 (ratio)
33554432 (n)   3046.333333 time (ms)   2.005486 (ratio)
67108864 (n)   6219.666667 time (ms)   2.041689 (ratio)
134217728 (n)   13488.000000 time (ms)   2.168605 (ratio)
```

log-log graph using all the data points:



Reasoning:

The reasoning here is similar to algorithm 2 – once fed large enough data sets that took at least a few milliseconds to process, the ratios settled close to 2. But this time I had no issues with running out of memory. Additionally, the slope of the log-log graph is very close to 1 (0.97). Therefore I can confidently say that this algorithm is linear.