EstimateSecretAlgorithms2

Max Friedman

Explanation

Simple Explanation:

Essentially, my code will automate each algorithm in turn, then for refinement, run each algorithm four times, averaging out “n” and “F(n)” for each algorithm, then will calculate log base 2 of both variables. Finally, it will return a doubling ratio for each algorithm which will let us infer the runtime of the algorithm.

More Detailed explanation:

- Starting with algorithm 1, my code takes a number “n”, then doubles it all the way up to 500,000,000. It then records “f(n)”, or the runtime necessary for the algorithm to run input “n” in microseconds (except Algorithm 4, as it is measured in nanoseconds because runtime was too quick). To have more refined numbers, I ran each algorithm four times and averaged each data-point.
- Since it can take a long time to run sometimes, my code automatically will break and run the algorithm again if it takes longer than a specific amount of time (I chose 8 minutes). However, if the fourth go-around of an algorithm occurs, the next algorithm is run with the same instructions.
- Then, now that each algorithm has been run extensively, giving us each a refined “n”, and “f(n)“, I took the log (every log calculated is in base 2) of each “n”, and “f(n)“, so we can plot a log-log graph. (Data for “n”, “f(n)”, “log(n)”, “log(f(n))” is below.)
- Finally, to determine the runtime each algorithm, my code uses the following doubling ratio formula: $f(2n)/f(n)$. The result of each doubling ratio on each “F(n)” is then averaged out to a single doubling ratio total. (Further explanation is below)

(Every data point, including each algorithm’s doubling ratio, was printed and calculated in one go by my code harness.)

<u>**Algorithms**</u>

Explanation:

<u>There are two ways to find runtime:</u>

- Find the doubling ratio of f(n).
    - Formula: $f(2n)/f(n)$
    - Explanation: For all "n", except the last "n" (because there will be no data point at "f(2n)"), find each doubling ratio. Then average the number for refinement.
    - Example:

n	f(n)
16	2.5
32	5
64	10

      $f(2 * 16) / f(16) = 2$, $f(2 * 32) / f(32) = 2$

      Average = (2 + 2) / 2 = 2

      Runtime: <u>**Linear**</u>
    - Ratio key:
        - ~1 = constant,
        - ~2 = linear
        - ~4 = quadratic
        - ~8 = cubic
- Find the slope of two points on the log-log graph.
    - Formula: aN^b , b = slope
    - Example:

log(n)	log(f(n))
4	1.3
5	2.3
6	3.3

      X,Y vales: (4,1.3) , (5,2.3)

      Slope = 1

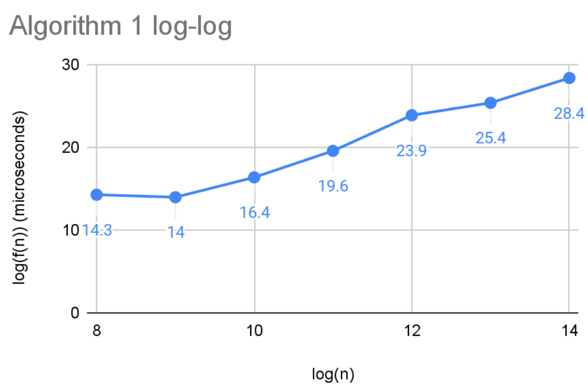
      $ax^1 =$ <u>**Linear**</u>

Each doubling ratio has already been found through my code harness. However, I will also show the slope method for redundancy.

Algorithm 1

Data and log-log Graph:

n	f(n) (microseconds)	lg(n)	lg(f(n)) (microseconds)
256	20352	8	14.3
512	16711	9	14
1024	83833	10	16.4
2048	780505	11	19.6
4096	1.59E+07	12	23.9
8192	4.35E+07	13	25.4
16384	3.47E+08	14	28.4



Doubling Ratio = 7.7

Explanation:

- Doubling Method:
  - Since the doubling ratio is ~8, our algorithm is <u>cubic</u>

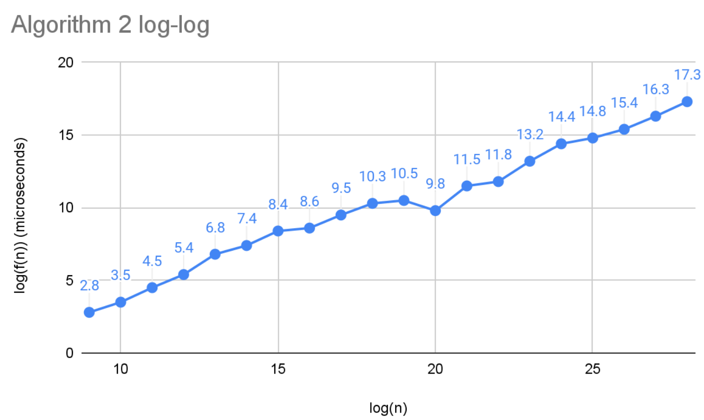
- Slope Method:
  - Two X,Y points on log-log graph: (13 , 25.4) , (14 , 28.4)
  - $\frac{28.4 - 25.4}{14-13} = 3$
  - $ax^3 =$ <u>cubic</u>

Final Answer: <u>Cubic</u>

Algorithm 2

Data and log-log Graph:

n	f(n) (microseconds)	lg(n)	lg(f(n)) (microseconds)
512	6.75	9	2.8
1024	11.5	10	3.5
2048	22	11	4.5
4096	42.75	12	5.4
8192	113.5	13	6.8
16384	166.25	14	7.4
32768	345	15	8.4
65536	389.75	16	8.6
131072	743.2	17	9.5
262144	1217.75	18	10.3
524288	1446.5	19	10.5
1048576	891.5	20	9.8
2097152	2814	21	11.5
4194304	3499	22	11.8
8388608	9408.5	23	13.2
16777216	21757.5	24	14.4
33554432	29092.25	25	14.8
67108864	43390.25	26	15.4
134217728	82834	27	16.3
268435456	163606.25	28	17.3



Doubling ratio = 2

Explanation:

- Doubling Method:
  - Since the doubling ratio is 2, our algorithm is <u>Linear</u>

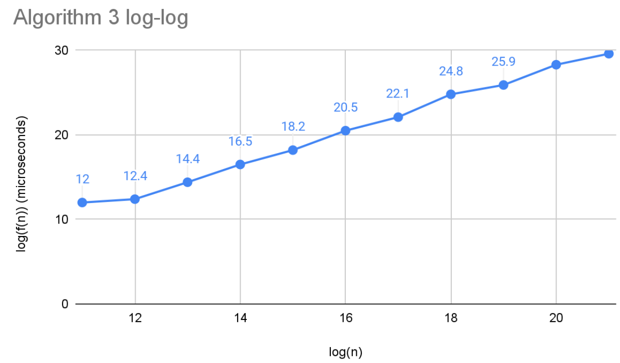
- Slope Method:
  - Two X,Y points on log-log graph: (26 , 15.4) , (27 , 16.3)
  - $\frac{16.3 - 15.4}{27-26}$ = ~1
  - ax^1 = <u>Linear</u>

Final Answer: <u>Linear</u>

Algorithm 3

Data and log-log Graph:

log(n)	f(n)	log(n)	log(f(n))
2048	3994.0	11	12
4096	5288.5	12	12.4
8192	20904.8	13	14.4
16384	94063.5	14	16.5
32768	292134.5	15	18.2
65536	1475187.0	16	20.5
131072	4628150.75	17	22.1
262144	2.96E+07	18	24.8
524288	6.17E+07	19	25.9
1048576	3.22E+08	20	28.3
2097152	8.40E+08	21	29.6



Doubling ratio ~4

Explanation:

- Doubling Method:
  - Since the doubling ratio is ~4, our algorithm is <u>Quadratic</u>

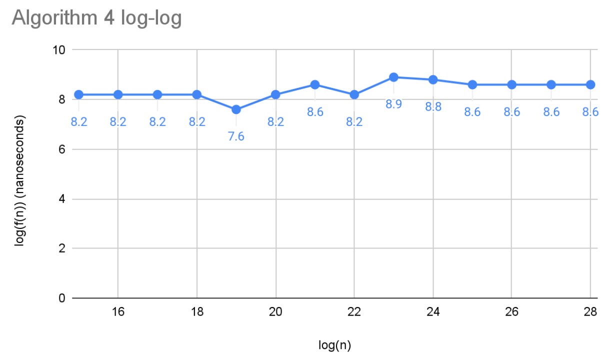
- Slope Method:
  - Two X,Y points on log-log graph: (13 , 14.4) , (14 , 16.5)
  - $\frac{16.5 - 14.4}{14-13}$ = ~2
  - ax^2 = <u>Quadratic</u>

Final Answer: <u>Quadratic</u>

Algorithm 4

Data and log-log Graph:

n	f(n) (nanoseconds)	lg(n)	lg(f(n)) (nanoseconds)
1024	299	10	8.2
2048	299	11	8.2
4096	301	12	8.2
8192	301	13	8.2
16384	200	14	7.6
32768	300	15	8.2
65536	300	16	8.2
131072	300	17	8.2
262144	300	18	8.2
524288	200	19	7.6
1048576	300	20	8.2
2097152	400	21	8.6
4194304	300	22	8.2
8388608	500	23	8.9
16777216	500	24	8.9
33554432	400	25	8.6
67108864	400	26	8.6
134217728	400	27	8.6
268435456	400	28	8.6



Doubling ratio ~1

Explanation:

- Doubling Method:
  - Since the doubling ratio is ~1, our algorithm is <u>Constant</u>

- Slope Method:
  - Two X,Y points on log-log graph: (25 , 8.6) , (26 , 8.6)
  - $\frac{8.6 - 8.6}{26-25} = 0$
  - ax^{0} = <u>Constant</u>

Final Answer: <u>Constant</u>