Homework - Estimate Secret Algorithms 2

Adam Frenkel

Description of the test harness:

In the following paragraphs I will provide a description of the code that I wrote in the class EstimateSecretAlgorithmsClient2.

At the top of the class there are four methods to run each of the algorithms. Each method sets up and executes the program and calculates the amount of time in either milliseconds or nanoseconds (depending on the algorithm different units of time were necessary) that it takes to execute.

The main method begins by running each of the four algorithms (with the methods that were described above) for several minutes in order to give a buffer period to allow for the jit compiler to get involved.

After the buffer period the programs runs through each of the algorithms three times. Each algorithm runs with a starting value of 250 passed in which is continuously doubled until it reaches an appropriate size for the given algorithm. (The program also runs with a value of 125 passed in, but that value is just there to help calculate the rate of growth).

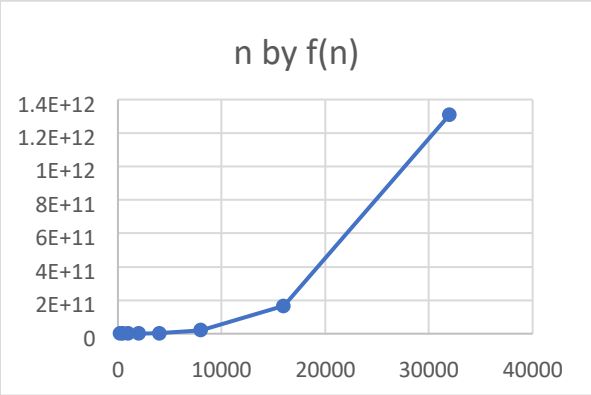
After all the algorithms are run three times, the program calculates the average run time of all the programs. It also calculates the rate of the growth (by dividing, for example, the run time of 1000 by, the runtime of the previous number, 500). Finally the program prints out all the data to be further analyzed by the programmer.

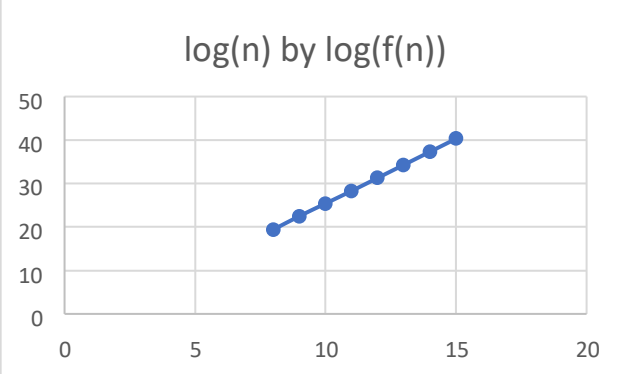
Over the next several pages the final results of this program are presented, and the growth rate of each algorithm is analyzed.

SecretAlgorithm1

Data:

n	f(n) Average Milliseconds	Rate of Growth	Log(n)	Log(f(n))
250	703791.0	7.6	8	19.4
500	5798292.0	8.2	9	22.5
1000	43545250.0	7.5	10	25.4
2000	338062167.0	7.8	11	28.3
4000	2629910417.0	7.8	12	31.3
8000	20817669417.0	7.9	13	34.3
16000	165540759417.0	8.0	14	37.3
32000	1310255500541.0	7.9	15	40.3





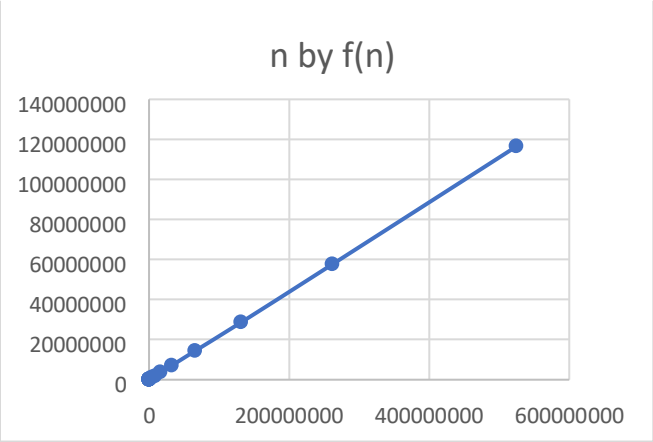
Analysis:

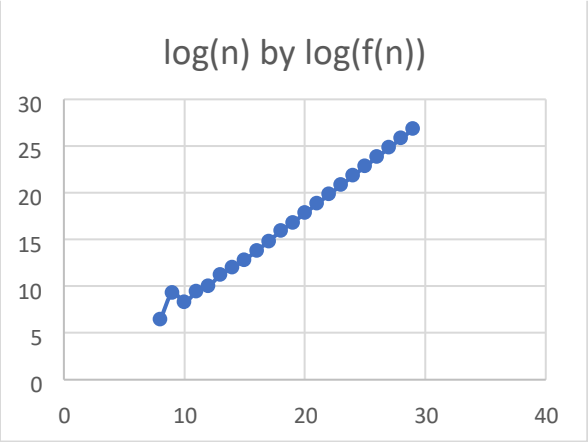
As the value of the input data (n) approaches a very large value (relative to the program), the rate of growth is clearly around 8.0. This means that to calculate the slope of the log-log chart we must solve for log8 (base 2), which of course is 3. This is clearly displayed in the log-log chart where there is almost consistently a slope of 3. Whenever the log-log chart of a program has a slope of 3 the program must be cubic. This leads to the conclusion that SecretAlgorithm1 has an order of growth that is **cubic** (i.e. n^3).

SecretAlgorithm2

Data:

n	f(n) Average Nanoseconds	Rate of Growth	Long(n)	Log(f(n))
250	83.0	0.3	8	6.4
500	625.0	7.5	9	9.3
1000	313.0	0.5	10	8.3
2000	688.0	2.2	11	9.4
4000	1000.0	1.5	12	10
8000	2334.0	2.3	13	11.2
16000	3958.0	1.7	14	12
32000	7167.0	1.8	15	12.8
64000	14750.0	2.1	16	13.8
128000	29084.0	2.0	17	14.8
256000	59708.0	2.1	18	15.9
512000	117917.0	2.0	19	16.8
1024000	234125.0	2.0	20	17.8
2048000	448875.0	1.9	21	18.8
4096000	900084.0	2.0	22	19.8
8192000	1783666.0	2.0	23	20.8
16384000	3595792.0	2.0	24	21.8
32768000	7135667.0	2.0	25	22.8
65536000	14332208.0	2.0	26	23.8
131072000	28537083.0	2.0	27	24.8
262144000	57704791.0	2.0	28	25.8
524288000	116406000.0	2.0	29	26.8





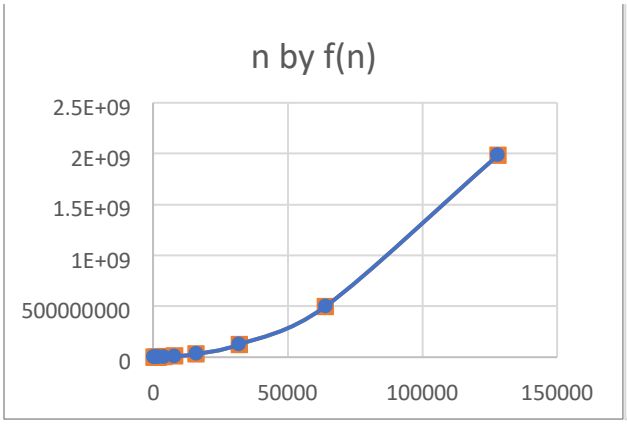
Analysis:

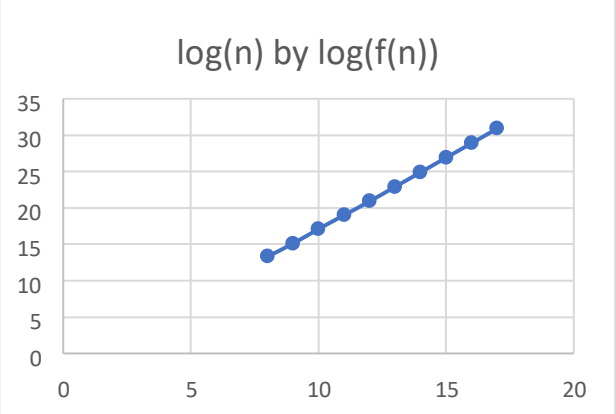
As the value of the input data (n) approaches a very large value (relative to the program), the rate of growth is clearly around 2.0. This means that to calculate the slope of the log-log chart we must solve for log2 (base 2), which of course is 1. This is clearly displayed in the log-log chart where there is almost consistently a slope of 1. Whenever the log-log chart of a program has a slope of 1 the program must be linear. This leads to the conclusion that SecretAlgorithm2 has an order of growth that is **linear** (i.e. n).

SecretAlgorithm3

Data:

n	f(n) Average Milliseconds	Rate of Growth	Log(n)	Log(f(n))
250	9792.0	0.2	8	13.3
500	35750.0	3.7	9	15.1
1000	135791.0	3.8	10	17.1
2000	522083.0	3.8	11	19
4000	2023000.0	3.9	12	20.9
8000	8014375.0	4.0	13	22.9
16000	31583542.0	3.9	14	24.9
32000	125436500.0	4.0	15	26.9
64000	498881167.0	4.0	16	28.9
128000	1983290291.0	4.0	17	30.9





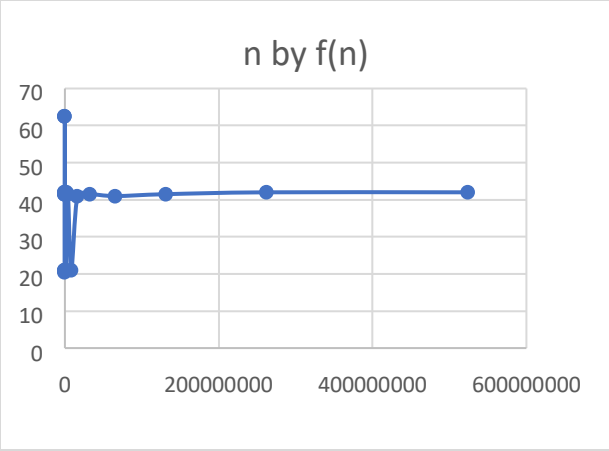
Analysis:

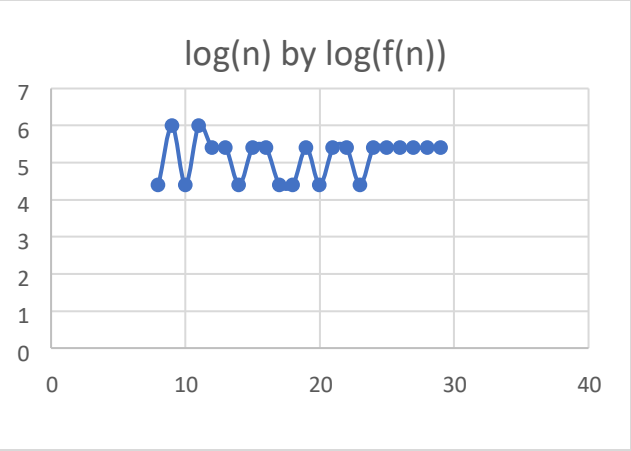
As the value of the input data (n) approaches a very large value (relative to the program), the rate of growth is clearly around 4.0. This means that to calculate the slope of the log-log chart we must solve for log4 (base 2), which of course is 2. This is clearly displayed in the log-log chart where there is almost consistently a slope of 2. Whenever the log-log chart of a program has a slope of 2 the program must be quadratic. This leads to the conclusion that SecretAlgorithm3 has an order of growth that is **quadratic** (i.e. n^2).

SecretAlgorithm4

Data:

250	21	1.0	8	4.4
500	62.5	3.0	9	6
1000	20.5	.3	10	4.4
2000	62.5	3.0	11	6
4000	41.5	.7	12	5.4
8000	41.5	1.0	13	5.4
16000	21	.5	14	4.4
32000	42	2.0	15	5.4
64000	42	1.0	16	5.4
128000	21	.5	17	4.4
256000	21	1.0	18	4.4
512000	42	2.0	19	5.4
1024000	21	.5	20	4.4
2048000	42	2.0	21	5.4
4096000	41.5	1.0	22	5.4
8192000	21	.5	23	4.4
16384000	41	2.0	24	5.4
32768000	41.5	1.0	25	5.4
65536000	41	1.0	26	5.4
131072000	41.5	1.0	27	5.4
262144000	42	1.0	28	5.4
524288000	42	1.0	29	5.4





Analysis:

As the value of the input data (n) approaches a very large value (relative to the program), the rate of growth is clearly around 1.0. This means that to calculate the slope of the log-log chart we must solve for log1 (base 2), which of course is 0. This is clearly displayed in the log-log chart where in the end there is almost consistently a slope of 0 (and in the beginning as well there are only slight deviations). Whenever the log-log chart of a program has a slope of 0 the program must be constant. This leads to the conclusion that SecretAlgorithm4 has an order of growth that is **constant** (i.e. 1).