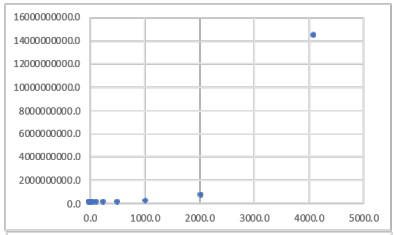
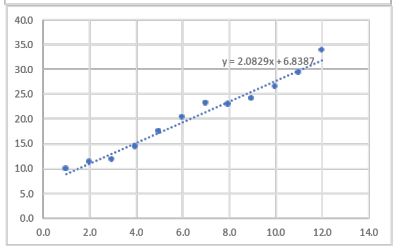
Arieh Chaikin – Assignment #2

I designed four similar methods to test various input sizes on each algorithm to see if I could determine the algorithms' order-of-growth. Each method consists of a "warm-up", which initializes the algorithm and runs it a few times to warm it up, as instructed. After that is complete, each algorithm is put through the main testing loop. This consists of, 1) Doubling the current input size, 2) Initializing the Algorithm with that input size, 3) Finding the current time of the system (in nanoseconds), 4) Running the Algorithm, 5) Finding the current time of the system (after running the algorithm), and 6) printing the difference between the time after running the algorithm and before running it. The data found, the logarithmic calculations used in my deliberations, and their graphs are presented on the following page.

Algorithm 1

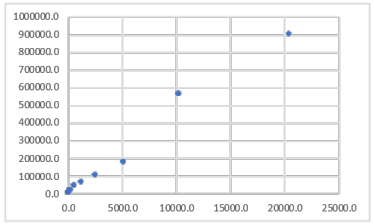
n	f(n)	log(n)	log(f(n))
2.0	911.0	1.0	9.8
4.0	2374.0	2.0	11.2
8.0	3369.0	3.0	11.7
16.0	21979.0	4.0	14.4
32.0	184906.0	5.0	17.5
64.0	1268843.0	6.0	20.3
128.0	9328735.0	7.0	23.2
256.0	8077209.0	8.0	22.9
512.0	17111767.0	9.0	24.0
1024.0	93461487.0	10.0	26.5
2048.0	629215978.0	11.0	29.2
4096.0	14361251860.0	12.0	33.7

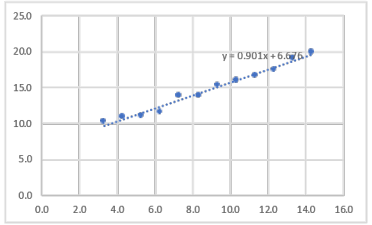




Algorithm 2

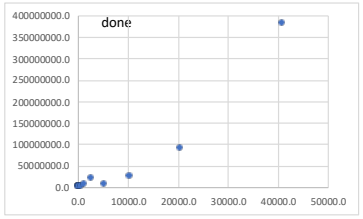
n	f(n)	log(n)	log(f(n))
10.0	1138.0	3.3	10.2
20.0	1974.0	4.3	10.9
40.0	2009.0	5.3	11.0
80.0	3050.0	6.3	11.6
160.0	14044.0	7.3	13.8
320.0	15005.0	8.3	13.9
640.0	40042.0	9.3	15.3
1280.0	64682.0	10.3	16.0
2560.0	101881.0	11.3	16.6
5120.0	173280.0	12.3	17.4
10240.0	563911.0	13.3	19.1
20480.0	900303.0	14.3	19.8

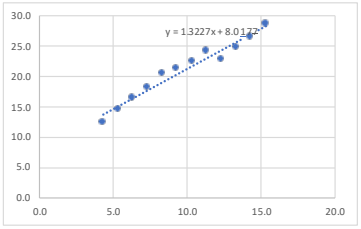




Algorithm 3

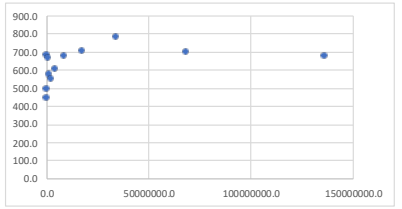
n	f(n)	log(n)	log(f(n))
20.0	5179.0	4.3	12.3
40.0	23522.0	5.3	14.5
80.0	88328.0	6.3	16.4
160.0	306947.0	7.3	18.2
320.0	1380906.0	8.3	20.4
640.0	2382853.0	9.3	21.2
1280.0	5533881.0	10.3	22.4
2560.0	19260396.0	11.3	24.2
5120.0	7262073.0	12.3	22.8
10240.0	26580092.0	13.3	24.7
20480.0	91997301.0	14.3	26.5
40960.0	380811515.0	15.3	28.5

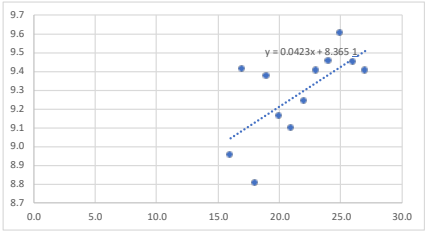




Algorithm 4

n	f(n)	log(n)	log(f(n))
66666.0	496.0	16.0	9.0
133332.0	680.0	17.0	9.4
266664.0	446.0	18.0	8.8
533328.0	662.0	19.0	9.4
1066656.0	570.0	20.0	9.2
2133312.0	547.0	21.0	9.1
4266624.0	603.0	22.0	9.2
8533248.0	676.0	23.0	9.4
17066496.0	699.0	24.0	9.4
34132992.0	777.0	25.0	9.6
68265984.0	697.0	26.0	9.4
136531968.0	676.0	27.0	9.4





Graph Row 1: f(n) Graph: X Axis measures n, Y Axis measures the runtime f(n) in nanoseconds

Graph Row 2: Log-Log Graph: X Axis measures log(n), Y Axis measures log(f(n)) (nanoseconds)

<u>Algorithm 1:</u> Quadratic. $O(N^2)$

The line generated by the log-log graph from the data I collected had a slope very close to 2, indicating a quadratic algorithm. After eyeballing both the data, and the f(n) graph (with its sudden increase), I concluded that this assumption was indeed likely.

Algorithm 2: Linear. O(N)

The line generated by the log-log graph from the data I collected had a slope just under 1, leaving me with to options, linear or linearithmic. Although in this type of experiment it is difficult to distinguish between these two types of graphs, linear graphs generally have a smaller slope than linerathmic graphs, so I concluded that this algorithm is likely to be a linear algorithm.

Algorithm 3: Linearthmic O(NlogN)

Similar to Algorithm 2, the line generated by the log-log graph from the data I collected had a slope around 1, this time above it, leaving me with to options, linear or linearithmic. With the same logic, I concluded that this algorithm is likely to be a linearthmic algorithm as their slopes are typically slightly larger than that of a linear graph.

Algorithm 4: Constant O(1)

The data collected from this algorithm indicated that regardless of the value of n, runtime would be somewhat constant (between 500 and 800 nanoseconds). Plotting the log-log graph confirmed by initial understanding with its line having a slope near 0.