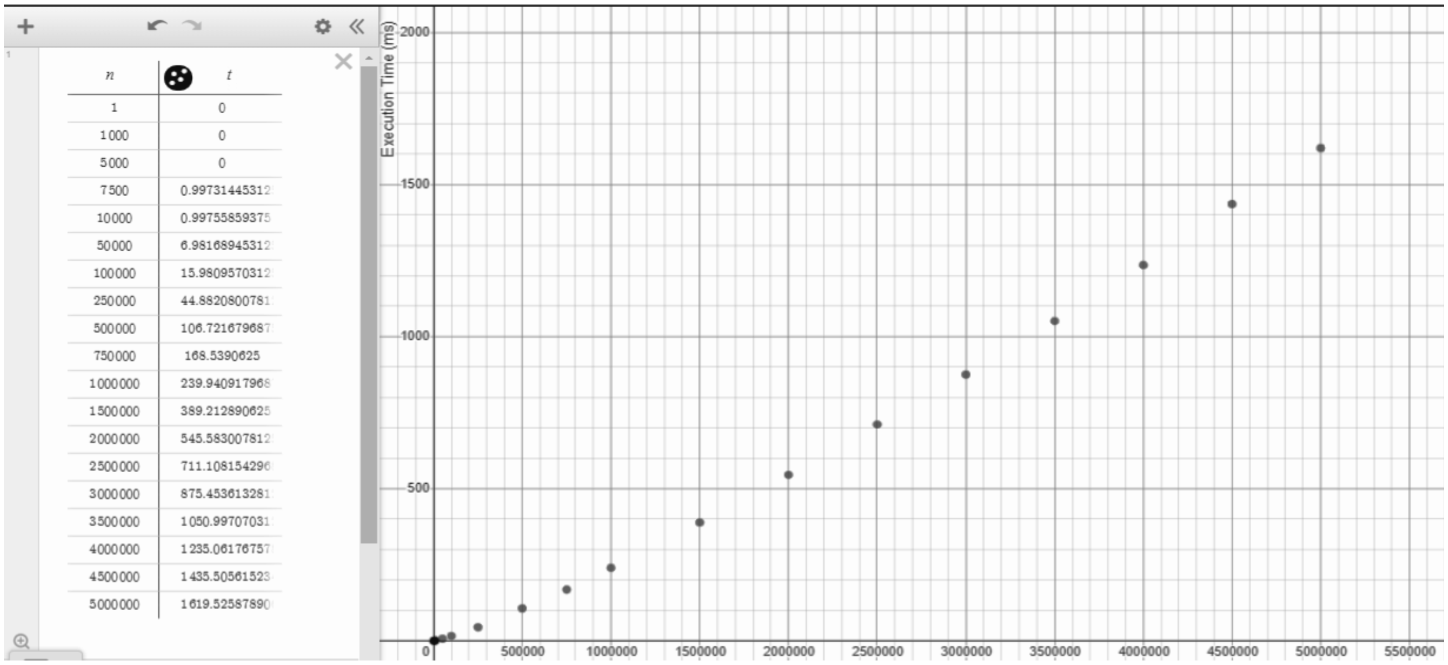
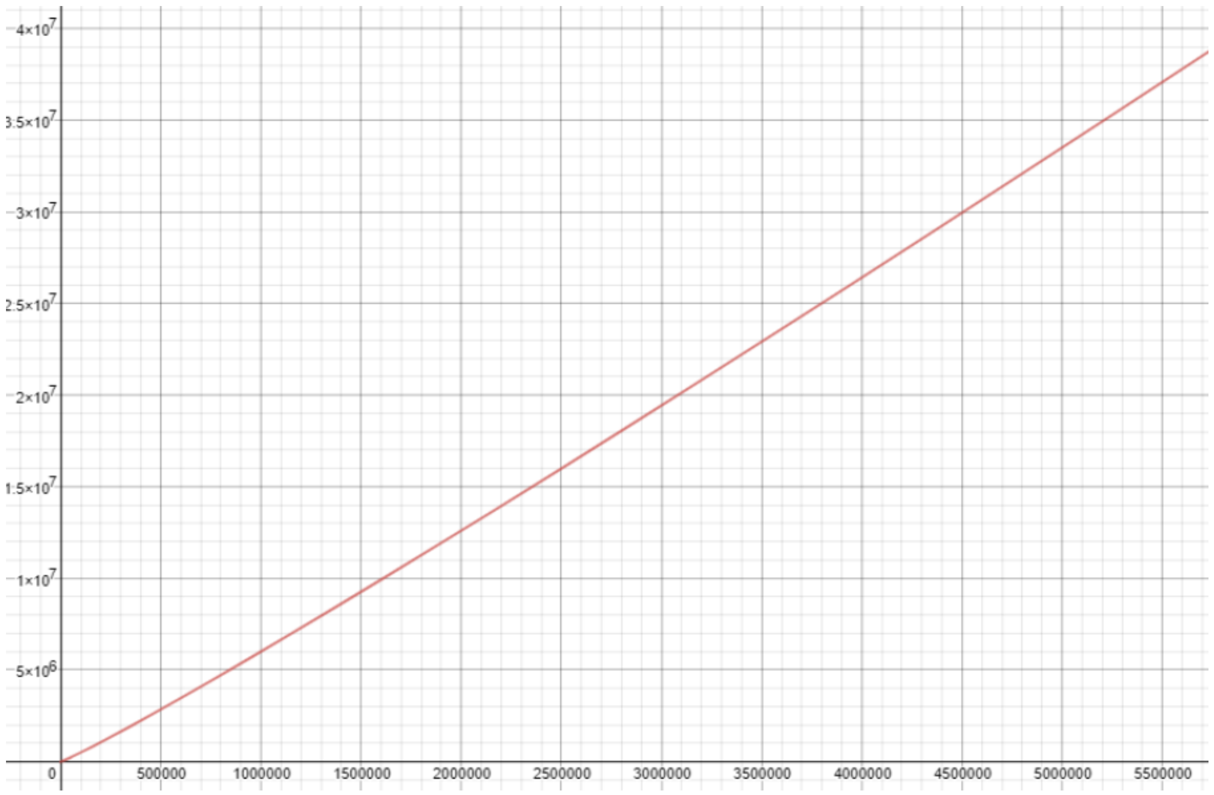
Problem 1:

Table of Data and Plot for sort-time.py code:



n	t
1	0
1000	0
5000	0
7500	0.99731445312
10000	0.99755859375
50000	6.98168945312
100000	15.9809570312
250000	44.8820800781
500000	106.721679687
750000	168.5390625
1000000	239.940917968
1500000	389.212890625
2000000	545.583007812
2500000	711.108154296
3000000	875.453613281
3500000	1050.99707031
4000000	1235.06176757
4500000	1435.50561523
5000000	1619.52587890

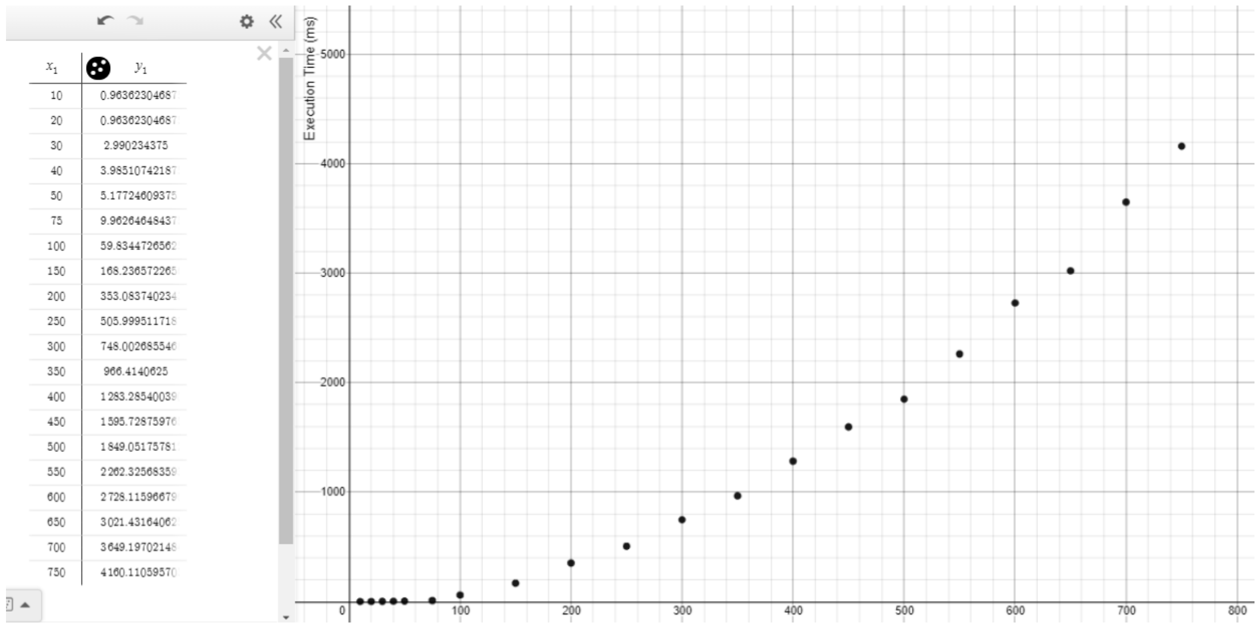
Plot of Upper Bound (n log n):



According to section 3.6, the sort() function has a Big-O efficiency and upper bound of O(n log n). Looking at the plot of sort-time.py and the upperbound (n log n), you can see a similar shape where it seems to grow a little slower with slower n values before growing faster with larger n values. They both look linear after a certain point, but with a zoomed-in view and being closer to zero, it is obvious that there is a curve for both. This confirms they are log linear or (n log n). The big difference is the scale of the graphs over the same interval. When trying to find a line of best fit for my data, 1/21000(n log n) fit the plot nicely. So, even though the scale of the graph is very different, they have similar growth curves.

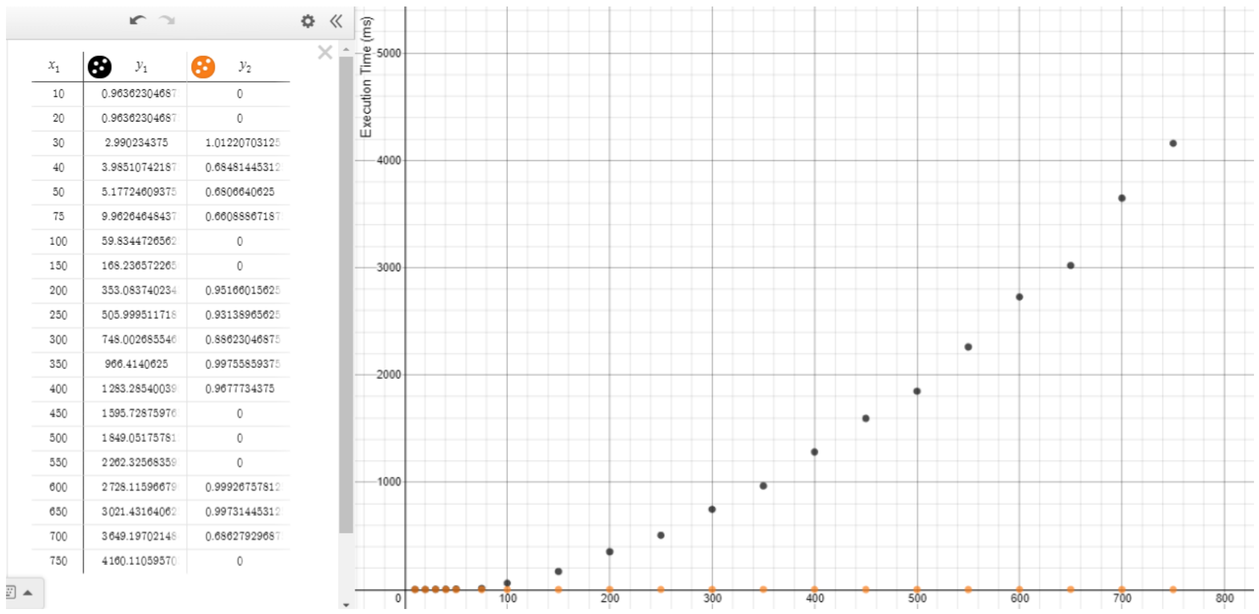
Problem 2:

Plot of Time Taken for Homework 3 Question 2 for n Sized List:



x_1	y_1
10	0.96362304687
20	0.96362304687
30	2.990234375
40	3.98510742187
50	5.17724609375
75	9.96264648437
100	59.8344726562
150	168.236572265
200	353.083740234
250	505.999511718
300	748.002685546
350	966.4140625
400	1 283.28540039
450	1 595.72875976
500	1 849.05175781
550	2 262.32568359
600	2 728.11596679
650	3 021.43164062
700	3 649.19702148
750	4 160.11059570

Based on my code, this takes O(n^2) time to execute. The plot shows this parabolic curve well in the image above. After seeing how bad the time went up I decided to refine the code to achieve the same result. My original code found every possible combination for 25 when you only needed to find one. The fix was easy and all I needed to change was putting a return function into the if statement. Included at the bottom of my code is my new function that runs much faster. Below, I have plotted both on the same graph to show the difference. Black dots for the original function and orange for the new function. I believe the time for my new function is logarithmic or O(log n).



x_1	● y_1	● y_2
10	0.96362304687	0
20	0.96362304687	0
30	2.990234375	1.01220703125
40	3.98510742187	0.68481445312
50	5.17724609375	0.6806640625
75	9.96264648437	0.66088867187
100	59.8344726562	0
150	168.236572265	0
200	353.083740234	0.95166015625
250	505.999511718	0.93138965625
300	748.002685546	0.88623046875
350	966.4140625	0.99755859375
400	1283.28540039	0.9677734375
450	1595.72875976	0
500	1849.05175781	0
550	2262.32568359	0
600	2728.11596679	0.99926757812
650	3021.43164062	0.99731445312
700	3649.19702148	0.68627929687
750	4160.11059570	0