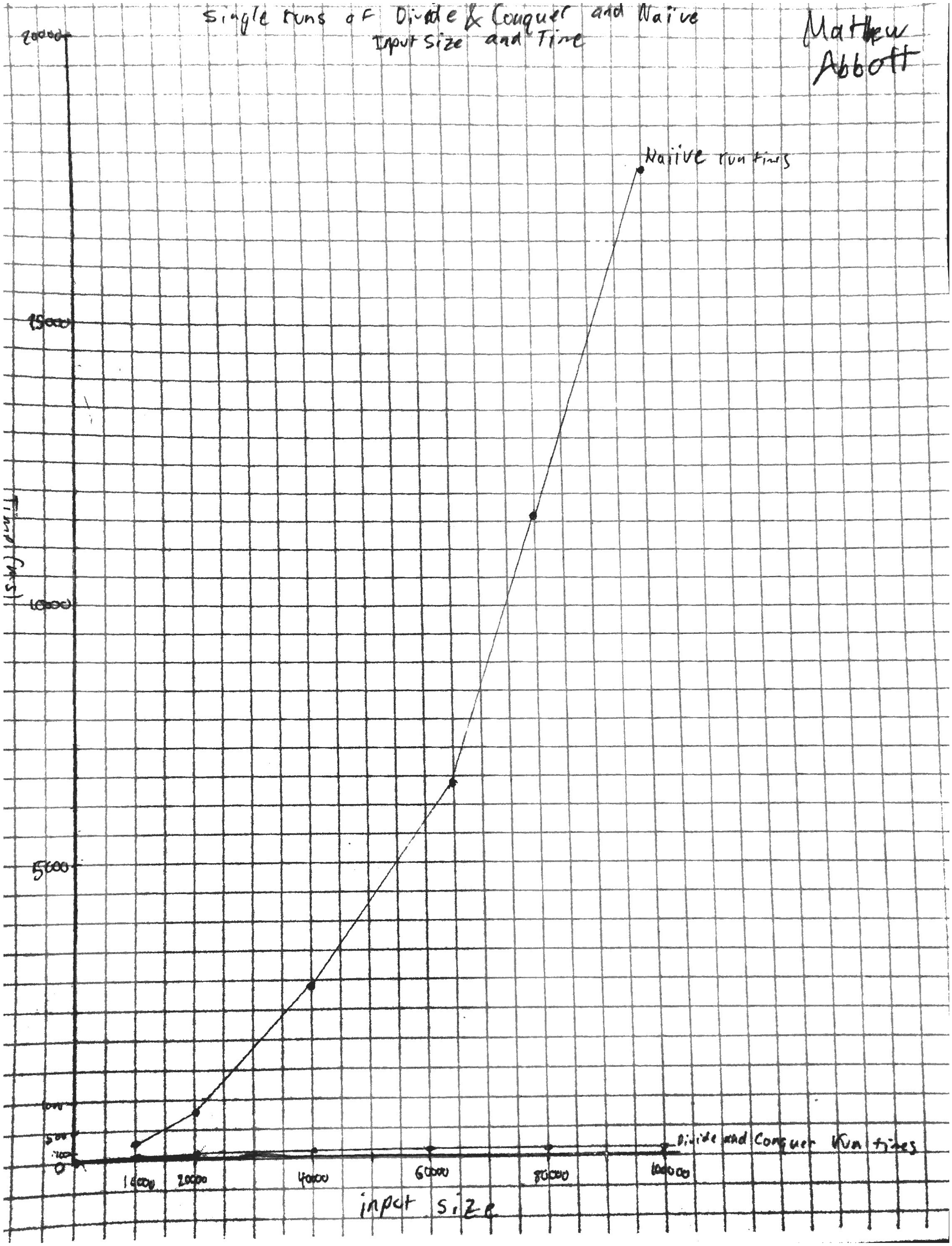
I did a basic implementation of the divide and conquer algorithm. It is as in the class handout except for two fairly innocuous things. The first is that I leave every distance as the squared value rather than the taking the square root until I've reached the final answer, since square roots are somewhat expensive and unnecessary up until the end. The second is that I store my coordinates in are array with my distance so that they can be returned by my recursive method. I discussed how to store the x and y coordinates of the points with several other people, and we came to the conclusion that this would be the simplest to implement, so I hope that any similarities on this specific line are ok. I never looked at anyone else's actual implementation of this idea, nor did I show my file to anyone else, I only engaged in discussion.

Upon experimenting, I found that the divide an conquer algorithm had nearly identical results when running the same input (of size 50000) 100 times, and 100 different inputs of that size. These were the results:

	min	**max**	**avg**
same run	16	109	35.74
different runs	15	110	34.81

At various large input sizes as outlined in the lab document, DC performed far, far better than the naive argument in a single run. Below is a table with the data and a corresponding graph.

size	**DC time (ms)**	**naive time (ms)**
10000	31	203
20000	46	751
40000	78	2937
60000	120	6478
80000	140	11687
100000	157	17753


Single runs of Divide & Conquer and Naive
Input Size and Time
Matthew Abbott
Naive run times
Divide and Conquer Run times
20000
15000
10000
5000
1000
500
100
0
Time (ms)
10000
20000
40000
60000
80000
100000
input size

To find the crossover point, I altered the main method to run the dc and naive algorithms 100,000 times with a different set of points each time (both algorithms used the same set of points though) and compare the average elapsed time for each. I then tried a number of different input sizes, trying to create upper and lower bounds for valid inputs. I wanted to find an input size such that dc was never slower than naive, but a smaller input would result in dc being faster. Once I found an input that seemed right, I tested it multiple times to see if it would fail at any point. I ended up finding that 155 was likely the most reasonable crossover point after I tested it 10 times and and the dc was barely faster every time. Below is a table of average run times over 100000 sets of points.

input size	**DC avg run time (ms)**	**naive avg run time (ms)**
130	.03146	.02961
135	.03449	.03221
137	.03228	.03357
137 (again)	.03798	.03073
138	.03108	.03618
138 (again)	.03599	.03432
139	.03546	.03379
140	.03282	.03743
140 (again)	.03406	.03482
140 (3rd time)	.03348	.03491
140 (4th time)	.03497	.03593
140 (5th time)	.03826	.03145
144	.03562	.03897
144 (again)	.03718	.03599
148	.03679	.03922
148 (again)	.03939	.03776
150	.03892	.039
150 (again)	.03758	.03806
150 (3rd time)	.03796	.03831
153	.03778	.0403
153 (again)	.04247	.03744
154	.03912	.04356
154 (again)	.03914	.04167
154 (3rd time)	.03982	.04177
154 (4th time)	.03931	.04182
154 (5th time)	.03893	.04181
154 (6th time)	.04029	.04177
154 (7th time)	.04061	.04275
154 (8th time)	.03794	.04306
154 (9th time)	.03899	.0405 (faster than .04061)
155	.03892	.04172
155 (again)	.03849	.04256
155 (3rd time)	.03894	.04235
155 (4th time)	.03701	.04415
155 (5th time)	.04011	.04154
155 (6th time)	.03783	.04408
155 (7th time)	.04036	.04174
155 (8th time)	.04021	.04225
155 (9th time)	.03726	.0417
155 (10th time)	.04001	.04133
156	.03861	.04573
160	.0404	.04553
160 (again)	.04241	.04457
180	.04391	.06064