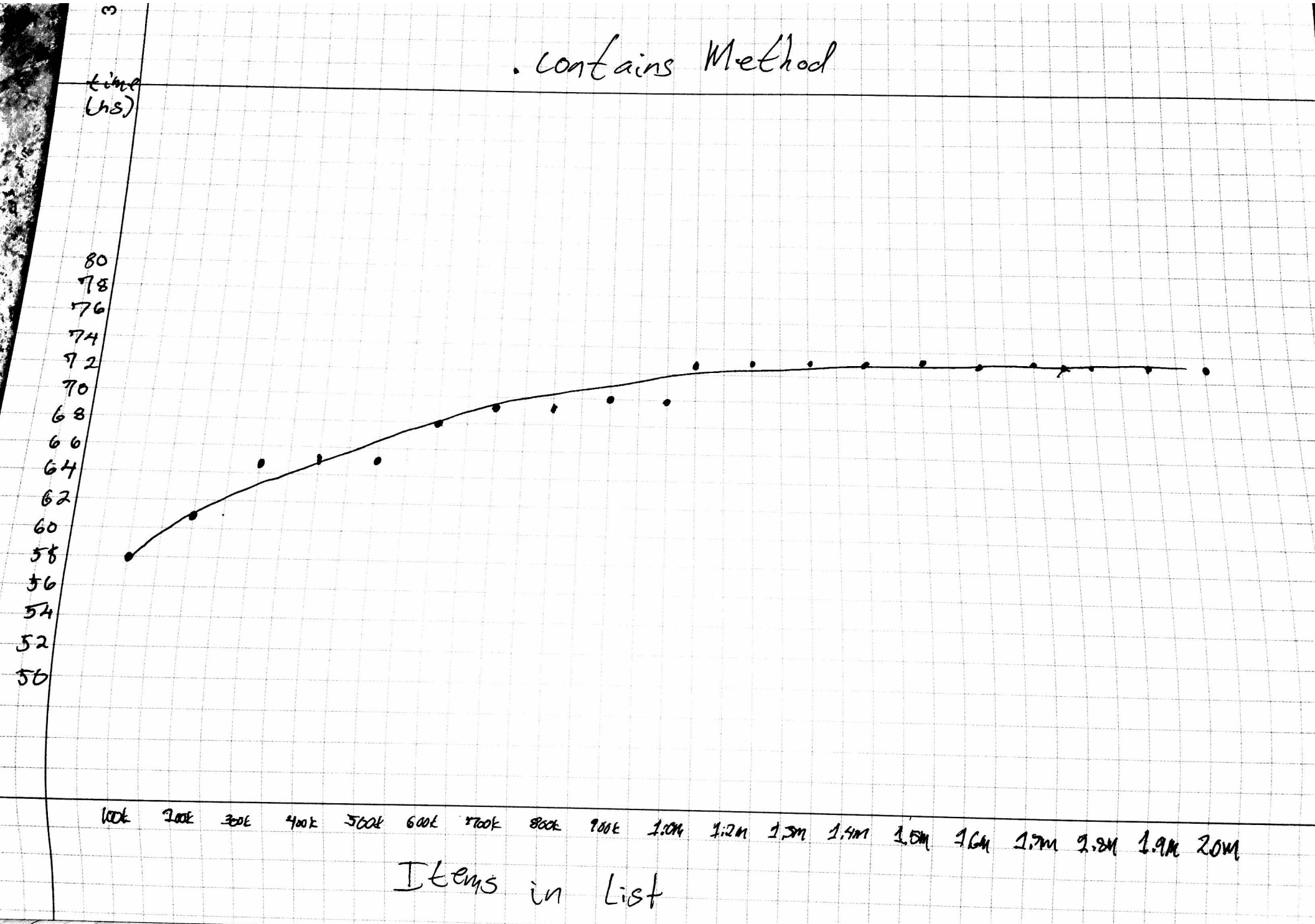
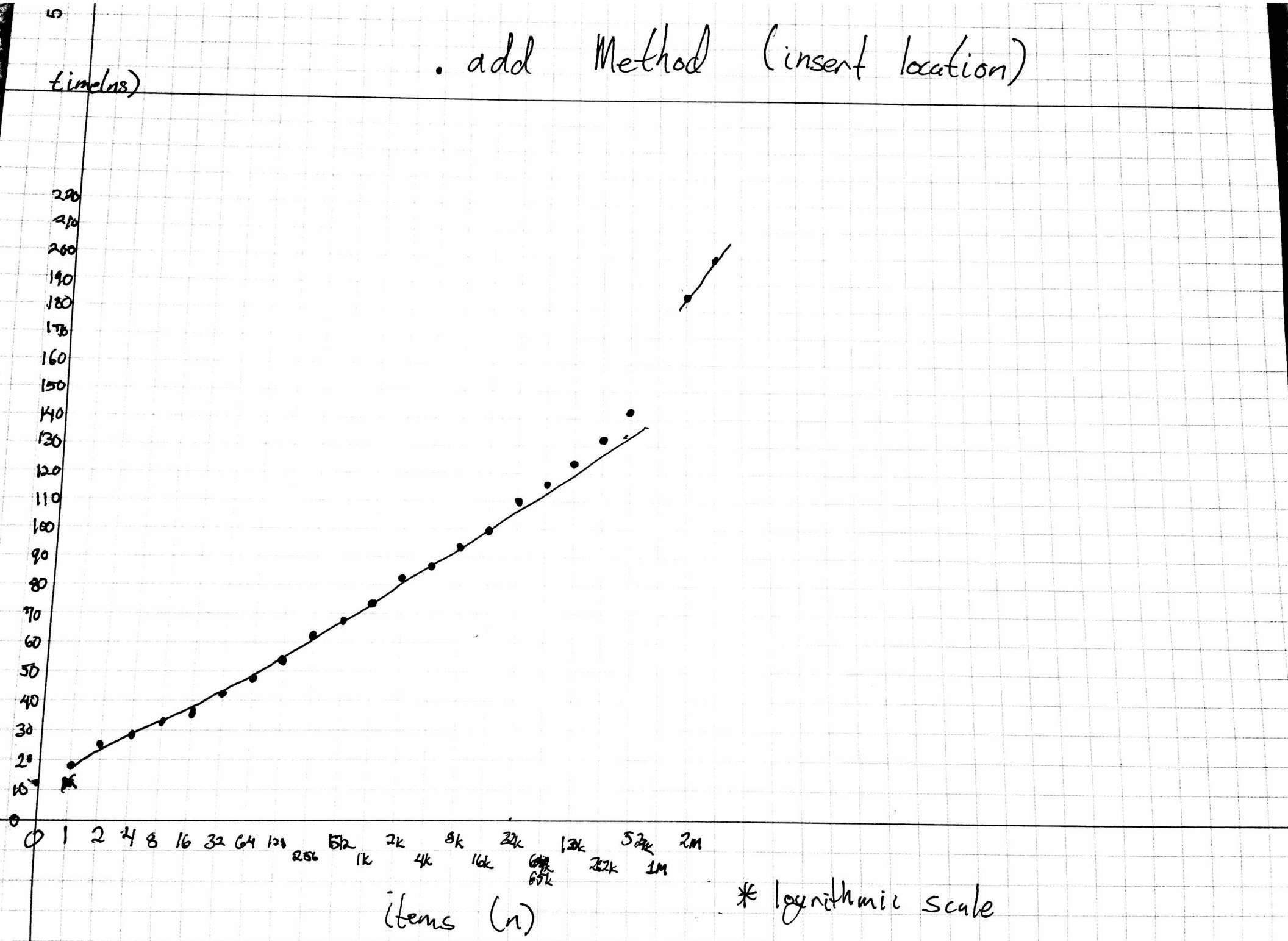

.contains Method
time (ms)
80
78
76
74
72
70
68
66
64
62
60
58
56
54
52
50
100k
200k
300k
400k
500k
600k
700k
800k
900k
1.0M
1.2M
1.3M
1.4M
1.5M
1.6M
1.7M
1.8M
1.9M
2.0M
Items in List

. add Method (insert location)



* logarithmic scale

Dyllon Gagnier

with Andrew Katsanevas

CS2420

Assignment 3 Analysis Document

1. My partner was Andrew Katsanevas and he turned in the code.

2. We switched roles about every hour at first and then about every twenty minutes later on in development when we were figuring out bugs. I actually proffered navigating this time because I was better able to see things at a distance from the keyboard.

3. I would definitely work with Andrew again. He's an excellent programmer and works hard.

4. To use a Java List data structure to back up our class would have had the advantage of not having to deal with growing the data structure manually, but would have had a number of developmental drawbacks. An array is easy to think about and accessing a value of an array is straight forward while random access for a List involves a call to a method which is a bit more typing and has more room for error. Additionally, the performance of this class could vary widely based on the particular implementation of the list interface. If, say, it were a linked list, then the random access required for binary search would make it such that it might be preferable to actually do a linear search since there is no true random access for a linked list. However, an array list would likely provide similar performance to the current implementation since our underlying data structure is essentially a specialized array list.

5. The algorithmic complexity of contains is undoubtedly O(log2(n)) because the basis of it is a binary search. It halves range of values that the potential element might be in by half each iteration until it either finds the element or the range of possible values is zero which is clearly logarithmic time.

6. My guess appears to have been correct. It was difficult to measure since the times are logarithmic, but it appears that the time to execute the algorithm increased by about 3-4ns per doubling of the size of the list, which makes sense since that requires one extra unit of work.

7. Locating where to insert an item not already in the list takes 2*log2(n) or log2(n^2). This is because it first must determine whether the item already exists in the list using .contains method which is log2(n) and then it performs a binary search to find the position where it should go which of course takes log2(n) operations where n is the size of the list. This is backed up by the experimental data where it takes about an extra 6ns for every doubling of n which is about twice the time of the .contains method which only performs binary search once. Strangely, for large n in the millions range, there was a sudden jump in execution time, but it otherwise appeared linearly when plotted logarithmically indicating a logarithmic function.

8. We spent approximately seven hours on this homework assignment.