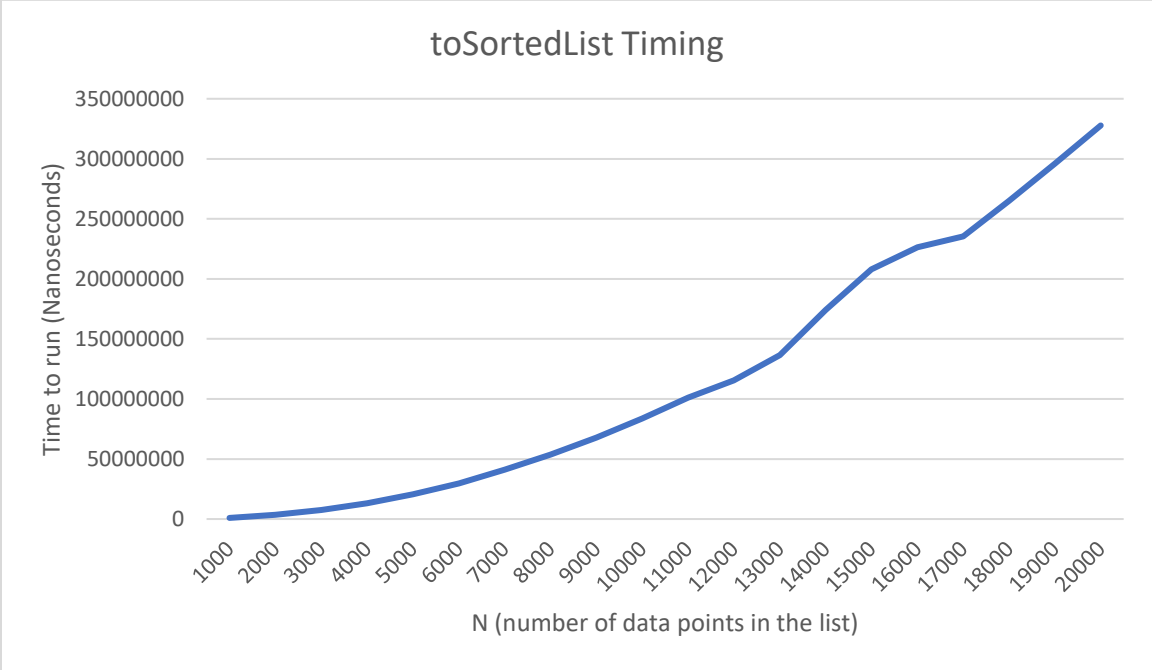
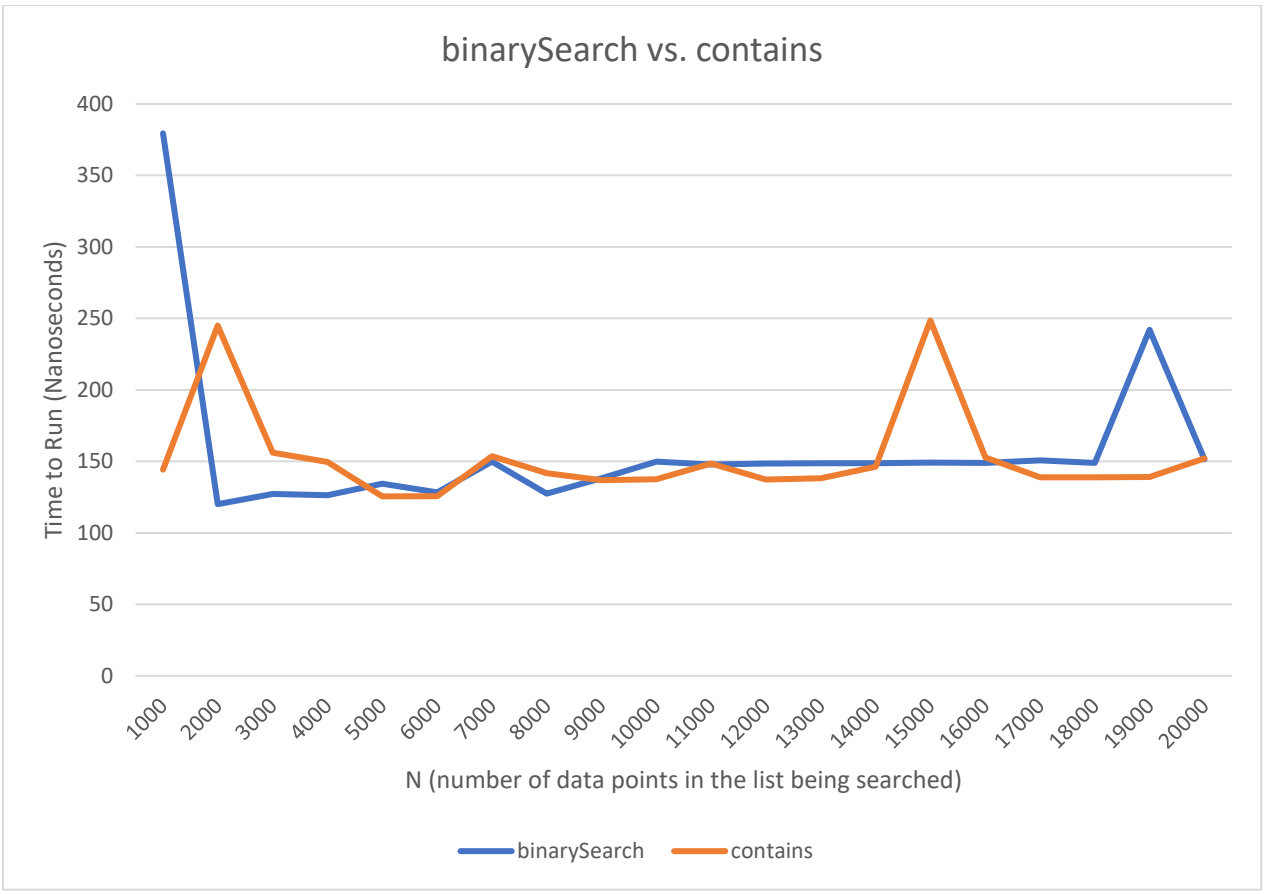
Assignment 3 Analysis:

1. My partner and I did very well with this assignment. We were able to resolve our issues with switching roles by working exclusively on Zoom, and we had no difficulty connecting this time. I think we both equally contributed to this assignment and communicated well. We also worked to create a more efficient code for this assignment, spending more time brainstorming coding ideas so we could have a better complexity. I also think we were better about trying each other's code ideas to see which was better.
2. I think the hardest part of this assignment was just the testing, because exhaustive testing required a large variety of tests, and we had to redo them all with a different Type than our original tests to ensure the generic nature of our code. It just took a lot of time to do, but it was very helpful because we noticed errors in a few of our methods that we would not have caught without all the tests.
3. 

4. 

5. The expected Big-O complexity is N^2 because we have a for-loop that loops N times, and within that for-loop we have another for-loop that runs N-1 times. When we multiply this together, we get N^2-N, but we also have another for-loop to create the sorted array that runs N times. So adding N to N^2-N we get N^2. The graph in question 3 supports this because in general, it follows a quadratic curve. There are only one or two data points that don't follow that curve, but they are barely off, and this is more than likely due to random background processes impacting our timing.
6. The Big-O complexity for our binarySearch method should be logN because we use one for loop that runs logN times. Apart from the first and last couple data points, it generally supports this expectation because we can see small curve that faces downward from 2000 N to 18000 N. However, when we include the first and last few data points, it no longer looks like a logN graph. This could be due to a spike in background processes while running the timer, but these points do make me question the actual Big-O complexity of our algorithm. I don't believe these spikes could be due to added method calls as the only method calls in the binary search loop is when we use get() to get a certain number, and this is consistent across all sets so that shouldn't cause such a big spike.
7. The worst case performance for contains is N, because in the method we loop over each value in the ArrayCollection, so if the desired integer is not in the ArrayCollection then the computer must still loop over every value before it can return false. The best case performance is 1, because if the desired integer is in the first spot, the loop only has to run once before it returns true. The average case would be N/2, because if the best case is

the lowest possible case and the worst case is N, then the average should just be half of the worst case which is N/2. I think our timing experiments are biased towards the worst case, because the probability of getting one integer out of two billion possible integer values is a very low percentage. If I did the calculations correctly, it's only about .01 percent. If we were biased towards the best case scenario, the possible integer range would be much smaller (for example, only integers 0-9) and our percentage would be more like 10%. Additionally, if we wanted a better percentage to bias our experiment towards the best case, we'd do better with a larger data set because there would be more chances for the random integer to be included in our array. As it is, the largest data set only has a .01% chance of having that integer, so it is clearly skewed towards the worst cases.