Assignment 3 Analysis

I (Chase Yates) worked again with Liam Astin on this assignment and I submitted the code. For this assignment we did mostly the same thing as assignment 2, I was the driver and Liam helped. Liam did a fine job helping out and contributed as best he could, we got our technical issues out of the way for good at the start of our session working together, as well.

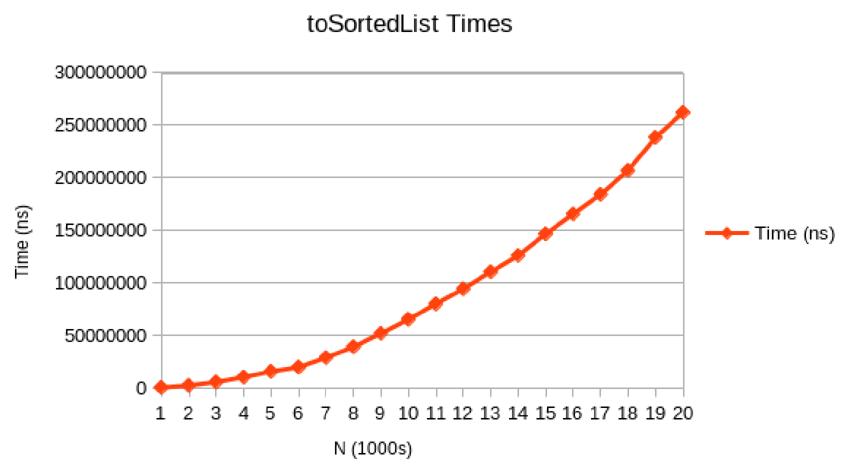
The most difficult part of this assignment for us was debugging our code after writing the primary code and our tests and running them. A very large number of our tests initially failed, but we worked through them eventually. The sorting algorithm in particular had us pulling our hair out at one point (It looked nearly identical to how it should've, just missed a +1 somewhere). We also realized early on that we were sorting in place when we should have been returning a new sorted set.

Experiments

We ran 2 timing experiments for this assignment.

Experiment 1

The first was to see the tendency of time it takes to run the `toSortedList` algorithm as the size of the list increases. For reference, the `toSortedList` method uses selection sort, creating a new sorted set. My hypothesis is that the graph of times will grow quadratically as N, the size of the list, increases. Here is the graph generated by our test:

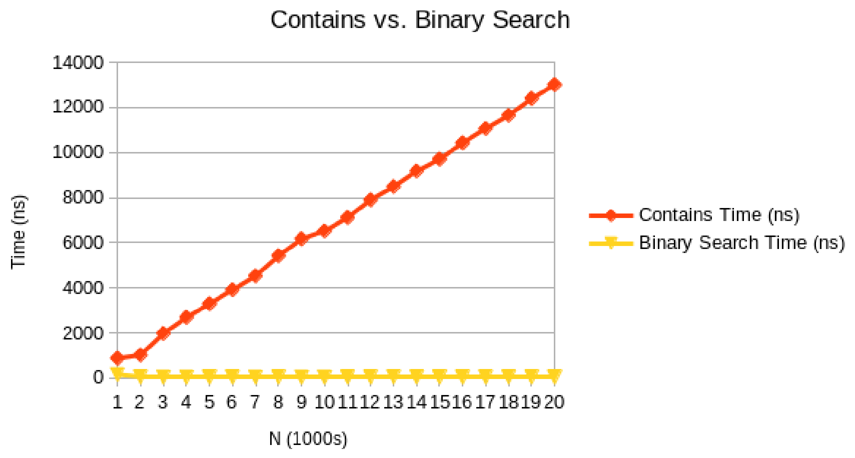


It seems to me that my hypothesis was correct. I ran this experiment with N going from 1,000 to 20,000 as indicated in the chart, getting average values after looping

2000 times. This took several minutes.

Experiment 2

The second experiment was similar, however it tested times of our contains methodas compared to our binarySearch method for finding items in our collections. My hypothesis is that the two algorithms are of two different complexity classes, the contains method being much slower than the binary search. Below is the graph of the result:



As you can see my hypothesis was right. The contains method runs incredibly slower than the binary search and seems to be linear.

Complexity

toSortedList/Selection Sort

Our `toSortedList` method uses selection sort. In studying the code, on every iteration as we run through the loop, it runs through the rest of the sub-array. This leads me to believe that selection sort is order N^2 / O(n^2). The graph supports this theory as it grows quadratically.

Binary Search

Our `SearchUtil.binarySearch` method binary searches through an `ArrayList` which I believe to be a logarithmic function/ O(logN). Unfortunately our experiment only suggests this, not confirming nor denying, as the chart shows what looks to be constant time. This is likely not true however as the chart is only so low because it is much outclassed by the seemingly linear `contains` method and is not really shown on the graph. The problem sizes we ran against the binary search also dont help as a logarithmic function needs much larger problems sizes to see its full growth over time

Contains

The contains method has a best case complexity of O(1) when the element you are looking for is the first in the array. The average case is O(N) as it must partially go through the array and as N increases the further the contains method must travel. The worst case is O(N) as well when the item isn't found and the method goes through the entire array. On every run of our contains method in this experiment, we were probably running the worst case as the number of possible numbers to be found is 2^31 whereas the largest size of the set is 20,000. 2^31 is vastly larger than 20,000 so we likely never even found the element we were searching for in this experiment, running us a cost of O(N) on every call to `contains`.