Alfred Neufeld
CS 2420 SP 15
2/4/15
Assignment 3 Analysis Document

I worked together with Jacob Bullard and I Submitted the source code. My programming partner and I switched roles approximately every thirty minutes. I think that this was appropriate amount of time. I feel that personally when I am the driver I can get sucked into what I am writing. Thus switching to a more broad role every thirty minutes keeps the big picture in the forefront of my mind. Jake was awesome with good ideas and an efficient work ethic I absolutely intend to work with this partner again. My partner and I spent roughly 10 hours on this assignment between debugging, testing, and actually writing code.

Using an internal Java list type of object instead of a basic array. Would have dramatically cut down our in program development time. This would mostly be due to use not needing to actually perform the nitty gritty work involved with growing the array, adding, or removing elements. We would simply be able to call the methods already associated with whatever type of list object we had selected to use. Certain details that exist in our current version of MySortedSet , such as the cursor integer used for maintaining indexing of the array would have not been needed. Overall I think that the time we would have saved in code writing would be well worth any run time trade offs that we would make.

Since contains is a binary search we expect the complexity to be of order O(log(n)). Derivation can be observed through the following steps.

It is known that the number of steps in a binary search can be represented by the following formula:

$$n\cdot\sum_{k=1}^{k=k_1}\left(\frac{1}{2}\right)^k$$

In order to determine the complexity we must determine the value of k_1

$$k_1\,occurs\,when \Rightarrow n\cdot\left(\frac{1}{2}\right)^{(k_1)}=1\Rightarrow n=2^{(k_1)}\Rightarrow \log_2(n)=\log_2(2^{(k_1)})\Rightarrow k_1=\log_2(n)$$

Thus we can conclude that a binary search has a big O complexity of O(log(n)). Observing figure 1 we can see that the plot demonstrates logarithmic behavior.

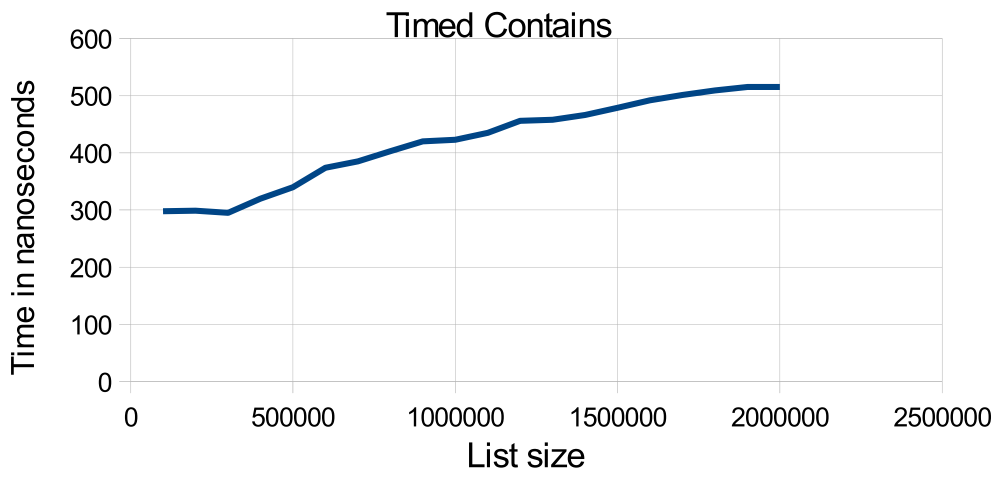

Figure 1. Graph detailing the time to list size relation for the contains method.

Due to how our code was set up our add method actually trended more towards O(n). This was due to the requirement that the list be maintained in sorted order thus in the worst case the program would be required to copy every element in the list over one place. Thus even though our binary search maintained logarithmic time in the add the function the dominant term would be linear one tending the over complexity of the function towards O(n). This can be observed in figure 2.

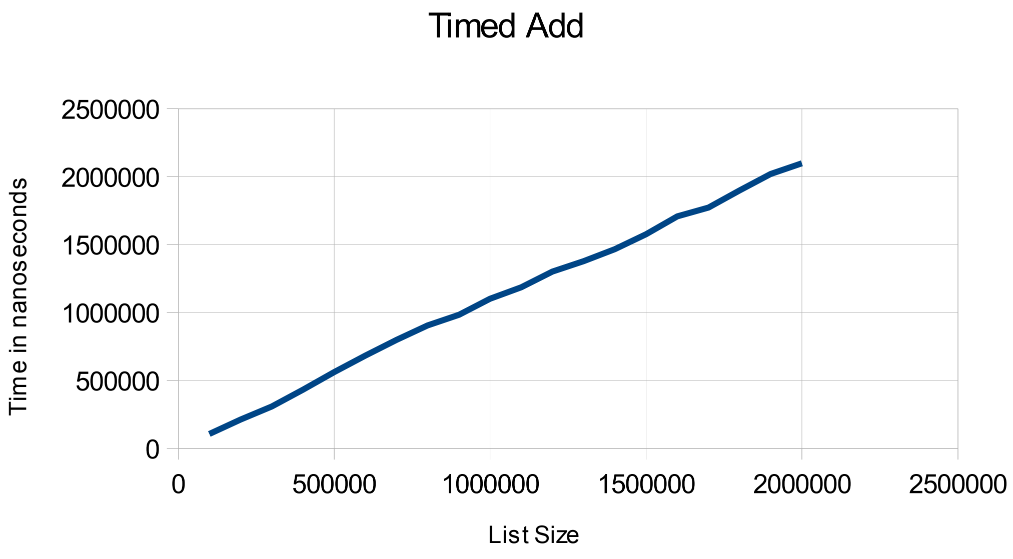

Figure 2. Graph detailing the time to list size relation for the add method

As seen in the graph the over time to list size relation is linear. This is due to the copying over of every data element for every insertion. I would consider this an implementation issue where the problem could easily be solved via use of a linked list where calling an insert function adds a constant overhead to a logarithmic binary search and your overall time complexity would stay in the realm of O(log(n)).