Phillip Pham
408-693-8252
October 6, 2014
EN.605.421.81.FA14
Programming Assignment #1

Programming Assignment #1

1a.) *Given this data structure, prove that the data structure yields O(lg n) time for searching but O($\sqrt{n}$) amortized time for adding an elements*

The data structure yields a O(lg n) time for searching because it uses the binary search methodology on two sorted lists (a short and a long list). This metholodgy identifies the minimum and maximum element of a list, calculates which index is the middle and compares it with the element that needs to be queried for. For this reason, it will always split the search space into halves, therefore, it will take O(lg n) time. Splitting the data into two sorted lists instead of one will not have a large effect on the complexity, therefore, will still take O(lg n)

The amortized time for adding elements is *O($\sqrt{n}$),* because of the fact that we have two sorted lists. The smaller list size is always going to be $\sqrt{n}$, so that is also the amoritized time for adding an element. The only caveat to using two sorted lists is that when the smaller list is full, it must be merged with the larger list. Because both lists are already sorted, it will only take N amount of time to merge them together, however, it only happens when the small list is full. With a very large N, the small list will be sufficiently big such that we will not have to worry about merges. For that reason, the amortized time of inserting an element will be the size of the smaller list, which is $\sqrt{n}$.

1b.) *Test your implementation to verify the predicted complexity bounds are achieved*

There are two input files that are used in the analysis (please see README for a description), a small and a large file. When looking at the trace logs, you can see that the binary search is O(lg n) for both the small inputs and large inputs.

Similarly, the amoritzed time for searching is also $\sqrt{n}$ by looking at the trace logs for both the small input and the large input. You can see that at the beginning of the insertions that there are many merges of the small and large lists, however, as the small list gets larger, the complexity will be the size of the small list because merges will not happen as frequently. For this reason, an amortized insertion time is $\sqrt{n}$.