README Sorted_List

Craig Perkins
Alex Tang

To implement a sorted list in C we decided to use a LinkedList. Each Node of the LinkedList contained a void* pointer to the data, a NodePtr to the next node and an integer to store the number of iterators pointing to the node. In total each Node took up at most 20 Bytes of data. When creating a SortedList we have a NodePtr to the front of the list. The SortedList also contains a pointer to a function and an integer that contains the size of the list. That means in terms of memory if we have a LinkedList with N nodes than the memory that a list will take up is O(N) or in total 20N + 8 + sizeof(comparison_function) Bytes of memory. The Iterator struct is also pretty efficient with memory. The iterator struct only contains a pointer to a Node in a sorted list. This means an iterator only takes up 8 Bytes of memory.

In terms of efficiency in order to insert a node into a sorted list the algorithm is done iteratively from the front node of the list. The means we have best case of O(1) and worst case of O(N). 4 compares are made during each iteration of the loop for a total of 4N comparisons in the worst-case. Because of the LinkedList implementation we unfortunately couldn't implement a kind of binary search insertion for O(logn), our algorithm runs in linear, O(N), time for insertion.

Our algorithm for deletion is also done iteratively from the beginning of the list. This means that best case the algorithm will be run in O(1) and at worst case will run at O(N). If we had more time we would have also liked to try to do a binary search kind of deletion, but we may have had to implement another kind of data structure for that. Another optimization that would be nice would be to check if the number being deleted was between the min and max of the list. If it is not we could be guaranteed that the item wouldn't be in the list, as of now our implementation goes through the whole list and searches every node. Our remove function takes into account if an iterator is pointing to a node and won't free up the memory right away if that is the case. It will wait for all iterators pointing to the node to move on before continuing to iterate.