Jesse Strivelli

ReadMe

This assignment create nodes that go into a linked list structures This linked list structure deals with opaque information (data) and inserts and removes this data while keeping the linked list sorted from largest being the head node to the smallest being the end. To traverse through the list you must create an iterator that starts to point to the head node of the list and contains a node pointer. In the best case for an insertion the linked list is either empty or the data contained in the node created is larger than the node sitting currently at the head which results in an O(1) case. In the worst case of an insertion the nodes data we have is smaller than all the nodes currently sitting in the list which results in a linear O(n) traversal. For the removing it is essentially the same thing. If the node that we want to destroy from the list is the first item, also the largest than we want we remove the head resulting in an O(1) case for removing. The worst case when removing data from the list would be that the information we are looking for is not in the list or the item we are looking for is the smallest item and last node in the list resulting in a linear O(n) traversal. I also have main that tests all the data in the linked list. It has comparator functions that deal with several different types of data. In the main we create different lists and perform insertions and removals and print the list out each time it is updated. This print runs in O(n) time in the best and worst case because we have to always traverse through the list in order to print it out. The secondary methods inside sorted-list.c such as SLCreate, createLnode, SLnodeDestroy, SLCreateIterator, SLDestroyIterator, SLGetItem, and SLNextItem all run in constant time O(1) because they are just returning values with no traversals. Lastly SLDestroy has a run time of O(n) because in the worst case the list is full. In the best case SLDestroy has an O(1) case where the list is already empty.