Chris Fretz
Systems Programming

Sorted List

- SLCreate
    - Not much going on here, function runs in constant time. O(1).
- SLDestroy
    - Function iterates across list and calls LNDestroy on each ListNode contained in the list. LNDestroy calls the destructor function passed into SLCreate, so, assuming the given destructor runs in constant time, SLDestroy runs in worst and best case O(n) time where n is the number of elements in the list.
- SLInsert
    - Function iterates across list until it finds a proper location to insert the given data. Worst case, element is the smallest, and it must iterate across all n elements, best case, element is the largest and it inserts immediately. Average case is probably something like o(n/2), which reduces to O(n).
- SLRemove
    - Function iterates across list until it finds an element matching the given data, at which point it removes the element from the list, and then, if there's an iterator parked on the node, it marks the node as having been removed from the list, otherwise it calls LNDestroy on the node. Both operations (assuming given destructor function runs in constant time) are constant, so runtime is based solely on iterating across the list. As with SLInsert, average case is probably something like o(n/2), which reduces to O(n).
- SLCreateIterator
    - Not much going on here, function runs in constant time. O(1).
- SLDestroyIterator
    - Even less going on here, function runs in constant time. O(1).
- SLGetItem
    - Even LESS going on here, function runs in constant time. O(1).
- SLNextItem
    - Function runs in constant time, simply hopping to the next node in the list and returning its data, unless the node it was parked at has been removed from the list. In such a case, function calls SLFindNext which iterates across the list until it finds a node that is smaller than the node SLNextItem was parked at, or it reaches the end of the list. Assuming a node is found, SLNextItem then calls LNDestroy on the old node if it was the only iterator parked on the node, hops to the node returned by SLFindNext, and returns its data. Otherwise, it conditionally destroys the parked node, and returns NULL. If parked node is in list, function runs in O(1), otherwise it runs in O(n). This unfortunately means that, if the parked node is removed from the list every time before SLNextItem is called, a full traversal of the list can run in, at worst, O(n^2) time.
- Memory Usage
    - ListNode struct contains 4 elements, 2 ints and 2 pointers. On the iLabs, this means that each node takes up 24 bytes. The SortedList struct contains 3 elements, all pointers, meaning that on the iLabs it also takes up 24 bytes. As such, a list itself will take up 24(n + 1) bytes (where n is the number of items in the list). The SortedListIterator struct contains 3 elements, one int and 2 pointers, so on the iLabs it takes up 20 bytes. So, all together, a list, plus its iterators, should take up 24(n + 1) + 20i where i is the number of iterators.