Ajay Srivastava (as1877) & Srihari Chekuri (svc31)

CS 214 Assignment 2 Readme: Sorted List

This program implements a sorted list by using a linked list. The iterator that was implemented accounts for the changes made to the list as it helps the user walk through lists. Since the objects in the list will be opaque, they are given as void* pointers. It was important to keep track of the nodes and to destroy any if they did not have any pointers to it. The "numberOfPointers" had to be incremented (or decremented) accordingly depending on the number of nodes pointing to a given node. As per the requirement, it avoids inserting duplicates into the sorted list. Also, we take into multiple corner cases such as running into one node that gets removed while having a pointer to it and such. Test cases are color coded to be in green and error messages (print statements for corner cases) are displayed with red to let the user know why the program returned a specific value.

SLCreate: First checks if the comparator and destruct functions are null. It allocates memory (using malloc) for the new, empty sorted list. It returns null if one or both of the functions are missing. Otherwise, it returns a SortedListt object. All this is done in constant or O(1) time.

SLDestroy: Destroys or frees each node until the list itself is destroyed (while the list is not null). Since there are n elements in the list, it has to iterate through n elements to completely free itself. Therefore, the big O for destroying a list is O(n).

SLInsert: After checking for nullity, it inserts the given object in the appropriate place in the list . It also allocates space for the new object, which is done in constant time. However, to actually insert the node, it has to go through n elements making the Big O of SLInsert O(n).

SLRemove: This function removes a given object from the sorted list after checking for the nullity in the list. It is important for the order to be maintained while removing the node while decrementing the pointer count. Similar to insert, the Big O for remove is also O(n) because it has to go through n elements to find the correct one so it can remove it.

SortedListIteratorPtr: This creates an iterator pointer which points to the first item in the sorted list if succeeded. We also allocated memory for the iterator pointer, making the run time Big O(1).

SLDestroyIterator: Destroys the iterator and then reduces the number of pointers by one. If the number of pointers pointing to a given node is 0, then we destroy the data in the node that was initially pointed to by the iterator and then free the node from memory.

SLGetItem: After checking for nullity, it returns the pointer data associated with the iterator pointer. This is also run in constant time big O(1).

SLNextItem: Returns the next object. Checks if the node is valid. If so, we allocate space and return data for the next object. If the node does not exist, we decrement the pointer count and if the count of the initial node is 0, then we destroy the data and then free the data.