Read Me

Run Time and Space Analysis of the following functions:

SLCreate: SL create mallocs the size of the void pointer and assigns the compare and destroy functions to each node created. This runs in O(1) time because it makes no comparisons and mallocs the size of an unknown object so that time cannot be accounted for. SLCreate takes up the space of the size of 1 void pointer with fields for comparing and destroying.

SLDestroy: SLDestroy runs in O(n+1) time which translates to O(n) time. This function travels to the end of the list freeing nodes one by one along the way. At the very end, upon reaching the last node in the list, it frees that node and then the list "head" which is passed into the function. This method successfully clears the entire structure created freeing it effectively without memory leaks.

SLRemove: SLRemove runs in O(n) time worst case where n is the number of nodes in the list. The worst case is when you are attempting to destroy the last node in the list. It has to do n comparisons to find the node and then set the node equal to null changing the previous nodes next.

SLInsert: SLInsert runs in O(n) time when n is the number of nodes already in the list. The worst case for insert is when you are inserting in the second to last spot. This is the worst case because you need to delete and free a node followed by doing two pointer changes. You need to change the nodes next's previous pointer to the node's previous and the nodes previous's next pointer to point at node next. Insert adds the size of 1 void pointer with each insert. The entire structure is the size of n void pointers plus one pointer from the list head to the first of the n void pointers.

SLCreateIterator: SLCreateIterator creates a pointer within the list to iterate through the list without changing what any of the pointers point to. Creating the iterator works in O(1) time and takes up the space of a void pointer.

SLDestroyIterator: SLDestroyIterator works in O(1) time and destroys the pointer used to walk through the list one by one.

SLNextItem: SLNextItem works in O(1) time and moves from the spot where the current Iterator is and returns the next item. This method is used to move through the list one by one.

SLGetItem: SLGetItem works in O(1) time and uses the node that SLNextItem left the iterator pointer at and simply accesses the item and returns what is stored in that particular node. This method is used in tandem with SLNextItem to move throught the list and access every item.