Systems Programming
Assignment 2: Sorted List
Efficiency analysis
Mark Conley && Michael Newman

SLCreate:

Creating a sorted list runs in constant time. SLCreate simply mallocs for the appropriate space and initializes relevant members of the sorted list struct. It runs in O(1) time and mallocs the size of the sorted list struct.

SLDestroy:

SLDestroy traverses the sorted list and frees nodes as it goes along. It runs worst case in O(n), where n is number of items in the sorted list. The relevant memory usage is it correctly frees the nodes in the sorted list.

SLInsert:

Worst case scenario for SLInsert is O(n), where n is the number of items in the list. This possibility happens when the object to be inserted is the smallest object in the list and must be inserted at the end. It always mallocs for the size of a new node but it frees the node if the object to be inserted is a duplicate.

SLRemove:

Worst case scenario for SLRemove is O(n), where n is the number of items in the list. This possibility happens when the object to be removed is the smallest object in the list and is therefore at the end of the list. If the reference count (number of pointer to the node) is less than or equal to 1 then we delete the node.

SLCreateIterator:

Creating a sorted list runs in constant time. SLCreateIterator simply mallocs for the appropriate space and initializes relevant members of the sorted list iterator struct. It runs in constant time and mallocs the size of the sorted list iterator struct.

SLDestroyIterator:

SLDestroyIterator runs in constant time. The function decreases the reference count and if the reference counts is less than one it deletes the node. Otherwise it frees the iterator.

SLGetItem:

SLGetItem runs in O(1). It runs in constant time because it just pointer to the data the iterator is currently pointing to.

SLNextItem:

SLNextItem runs in O(1). The function runs in constant time because it just increments the iterator to the next item in the list and returns its data.