SLCreate(): Creates a sorted list and allocates space and initializes everything inside it. This happens in O(1) time.

The head will only be created once, so the initial space will be 40 bytes (based on sizeof(SortedListPtr)).

SLInsert(): Takes a void* argument and inserts it (or doesn't) into the sorted list based on the comparator function. If the sorted list has a head whose data is less than the new object, equal to the new object, or if the sorted list has no head, then this function will run in O(1) time. This function will also run in O(1) time if the list only has one node, the head, and its data is greater than the new object. At the worse case SLInsert() will run in O(n).

The space complexity of SLInsert will be based on how many nodes are entered, 40bytes*n.

SLRemove(): Takes a void* argument and removes the node in the sorted list containing data that matches it. If the sorted list has a head whose data is less than the new object, equal to the new object, or if the sorted list has no head, then this function will run in O(1) time. This function will also run in O(1) time if the list only has one node, the head, and its data is greater than the new object. At the worse case, SLRemove() will run in O(n).

This will not reduce or increase the space complexity, the program will remove the node from the order of the sorted list, but until the node is overwritten, the data still exists.

SLCreateIterator(): Allocates space for an iterator if a list exists, and makes it point to the head. This function runs in O(1) time whether there is a list or not.

Each Iterator created will take up 8 bytes (according to sizeof(SortedListIteratorPtr)).

SLGetItem(): Gets the data at the node that an iterator is pointing to, if the node exists. This function runs in O(1) time whether the node exists or not.

No change in space complexity.

SLNextItem(): Gets the data at the node following the node that an iterator is pointing to, if that node exists. This function runs in O(1) time whether that node exists or not.

A node maybe destroyed reducing the space complexity.

SLDestroy(): Frees a sorted list in O(1) time.

SLDestroyIterator(): Frees an iterator in O(1) time.