Sorted List Notes and Big-O Analysis
Andrew Moore and Paul Jones

SLCreate

This creates the memory for an empty sorted list. The analysis depends on what the rest of the OS is doing, as it only allocates memory. It allocates space for the sorted list struct and head node pointer.

$$O(1)$$

SLDestroy

This frees all the memory for a list. It does this recursively, freeing the list one node at a time from the end of the list. After each and every node has been individually freed, it frees the sorted list struct.

$$O(n)$$

SLInsert

This inserts an object into the sorted list. Because we used a linked list, we could not use Binary Search ($O(log(n))$). This uses a simple linear search.

$$O(n)$$

SLRemove

This removes an object from the sorted list. As with SLInsert, we used a linear search to find the node requested for deletion. Note that unless the reference count for the node is 0, this method does not free the node. Instead, it leaves it around - it is the responsibility of the last referencer to clean up the memory. However, nodes that are still referenced (by iterators, we can assume) are still disconnected from the list by disconnecting it from the node before it.

$$O(n)$$

SLCreateIterator

This creates an object that allows the caller to iterate through the list. It allocates the struct and starts the iterator at the head.

$$O(1)$$

SLDestroyIterator

This frees the memory used for the iterator struct. The actual list is not touched.

$$O(1)$$

SLNextItem

Returns the next object from an iterator. While the iterator sits on an object, its item pointer is considered persistent and raises the relevant node's reference counter by one. This method also has the responibility of freeing a node who's only referencer was this iterator.

$$O(1)$$