Sorted-List – Written by Tom Milburn & Zachary Psaras

Big-O Runtimes are as follows with small justifications. All runtimes are in terms of worst-case unless noted otherwise.

SLCreate(): O(1). There are no loops. Initialization of the SortedList objects should be constant time.

SLDestroy(): O(n) where n is the number of list elements in the passed-in SortedList object.

SLInsert(): O(n) where n is the number of list elements before insertion. SLInsert traverses the list once, comparing every element's data with the new data object being passed into the function.

SLRemove(): O(n) where n is the number of list elements before removal. SLRemove traverses the list once, comparing every element's data with the data passed in.

SLCreateIterator(): O(1). There are no loops. Simple initialization of an iterator struct.

SLDestroyIterator(): O(1). Just one free statement.

SLGetItem(): O(1). Simply accesses member 'data' pointed to by the object referenced in the iterator object.

SLNextItem(): O(1). Simply sets the given iterator to the next list element, provided we are not at the end of the list. If the end of the list is found, iterator is set to NULL.

Memory efficiency is as follows with small justifications.

SLCreate(): Initializing a SortedList object malloc's once for the object itself. The object contains 3 pointers and an integer.

SLDestroy(): Initializes two pointers to Node objects.

SLInsert(): Malloc's a new node to be inserted as a SortedList element. Also initializes a few pointers.

SLRemove(): Initializes a pointer.

SLCreateIterator(): Malloc's memory for an iterator struct.

SLDestroyIterator(): No memory requirements outside of that defined by free() and whatever the function overhead is.

SLGetItem(): No initializations.

SLNextIte(): No initializations.