README

Artem Titoulenko
Jeffrey Adler

Big-O Analysis

struct SortedList* SLCreate(CompareFuncT cf);

Creation involves allocating space, and initializing values. This happens in constant time. *O(1)* for both memory and running time.

void SLDestroy(struct SortedList* list);

Destruction of `Node`'s is recursive and thus linear. This is done by calling `destroy_nodes` the next nodes before `free`'ing the current node. *O(n)* for both memory and running time.

int SLInsert(struct SortedList *list, void* newObj);

Insertion swaps the leading node with it's next node until the next node is greater than or equal to the current node. This is linear. *O(n)* for running time and *O(1)* for memory.

int SLRemove(struct SortedList *list, void* newObj);

Removal traverses every element in order until the comparison function returns true. This only deletes one instance of the object. This operation is linear. *O(n)* for running time and *O(1)* for memory.

struct SortedListIterator *SLCreateIterator(struct SortedList*

list);

Creating an iterator involves allocating space and initializing values. These things happen in constant time. *O(1)* for both memory and running time.

void SLDestroyIterator(struct SortedListIterator* iter);

Destruction of an iterator is a matter of `free` 'ing the memory. This is a constant time operation. *O(1)* for both memory and running time.

void * SLNextItem(struct SortedListIterator* iter);

This function gathers the current node's data, and then replaces the current node in the iterator with the next node. This is done in constant time. *O(1)* for both memory and running time.