CS214 Programming Assignment 2

Vincent Xie, Jacob Rizer

**Sorted list**

Our implementation of the sorted list uses a linked list. When we add to the list, we create new nodes, and when we delete from the list, we remove nodes. I will begin to explain our process using the structs that we used.

**struct Node**

- data (holds the data that the node holds)

- prev (holds reference to previous node)

- next (holds reference to next node)

- references (holds number of references to the node)

- removed (holds 1 if node is slated for deletion)

**struct SortedList**

- compare (holds the compare function that allows for two items to be compared)

- destroy (holds the destroy function that deallocates the memory used for data)

- head (holds the reference to the first node in the list)

**struct SortedListIterator**

- current (holds the reference to the node that the iterator is pointing to)

- destroy (holds the destroy function that deallocates memory used for the data)

- list (holds a reference to the list, needed for function correctIterator())

**Functions**

These functions are what the caller uses to interface with the sorted lists.

**Node \*createNewNode()**

Analysis:

Time: O(1)

Space: O(1)

Params: None

Return: Pointer to new node

Creates a new node by allocating memory and setting its fields.

**SortedListPtr SLCreate(CompareFuncT cf, DestructFuncT df)**

Analysis:

Time: O(1)

Space: O(1)

Params: Compare function, Destruct function

Return: Pointer to new list

Creates a new empty list. The head is allocated (not see to null) and the compare and destroy function pointer fields are set accordingly.

**void SLDestroy(SortedListPtr list)**

**int SLInsert(SortedListPtr list, void \*newObj)**

Analysis:

Time: O(n), where n = number of elements in the list

Space: O(1)

Params: Pointer to list, New object to add

Return: 1(successful), 0(unsuccessful)

Adds new node to the list by traversing the list and adding a new node where it would belong in the list in descending order. Does not take in duplicates, empty lists, or empty items (return 0).

**int SLRemove(SortedListPtr list, void \*newObj)**

**SortedListIteratorPtr SLCreateIterator(SortedListPtr list)**

Analysis:

Time: O(1)

Space: O(n), where n = number of elements in the list

Params: Pointer to list, New object to add

Return: 1(successful), 0(unsuccessful)

Adds new node to the list by traversing the list and adding a new node where it would belong in the list in descending order. Does not take in duplicates, empty lists, or empty items (return 0).

**void SLDestroyIterator(SortedListIteratorPtr iter)**

**void \* SLGetItem( SortedListIteratorPtr iter )**

**Node \*correctIterator(SortedListIteratorPtr iter, void \*item)**

**void \* SLNextItem(SortedListIteratorPtr iter)**