Arifur Rahman
Systems Programming
PA2

SLCreate(): This will create a sorted list and allocate enough space and puts data inside it. This happens in O(1) time. Since the front is only created once, the space is 40 bytes which I get after using sizeof(SortedListPtr).

SLInsert(): This will take a void* argument and insert it (or doesn't if it already exists) into the sorted list according to the comparator function. If the sorted list has a front whose data is less than the new object, equal to the new object, or if the sorted list has no front, 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 front, and its data is greater than the new object. Worst case will run in O(n).

The space complexity of SLInsert will match accordingly to how many nodes are entered, 40bytes*n.

SLRemove(): This will take a void* argument and remove the node in the sorted list containing data that equals the void* argument. There are three cases related to run time: when sorted list has a front with data less than a new object, when sorted list has a front with data same as the new object or when there isn't a front. Run time is O(1) for all these cases. Similarly if the list has one node only which is front and if its data is greater than the new object, run time will still be O(1). Worst case for this function is O(n).

The program will remove the node from the order in the sorted list, but before that node is overided, there will be some data that exists. Thus space complxity will neither increase or decrease.

SLCreateIterator(): This will allocate space for a n iterator (if the list exists) and makes it point to the front. This runs in O(1) time even if ther e isn't a list.Every iterator will use 8 bytes acc ording to sizeof(SortedListIteratorPtr))

SLGetItem(): This will get data from a node that a n iterator is pointing to. This runs in O(1) time even if the node doesn't exist. There isn't a chan ge in space complexity.

SLNextItem(): This will get some data at the node that comes after the node an iterator is pointing to. This runs in O(1) time even if the node doesn 't exist. Worst and Best case memory usage is O(1) since it is moving the pointer,not allocating mem ory.

To reduce space complexity, a node can be destroyed .

SLDestroy(): Frees a sorted list in O(1) time. Thi s has to free the empty front node and the SortedL ist struct. No additianal memory is needed to be a llocated when this is called.

SLDestroyIterator(): Frees an iterator in O(1) tim e.This has a fixed number of operations to perform which has nothing to do with the input size.