We thought linked list would be the most useful for this sorted-list assignment and therefore, we used it in order to handle changes, list and store the numbers. If an user requests next item from iterator, it is checked if the count matches first, then if it does it responds but if it does not then pointer points to the next valie in that list which is less thatn the last one.

Analysis

SLCreate:
Runtime: Allocate memory for SortedList and default values Constatnt O(1)
Memory: One more than SortedList struct Constant O(1)

SLInsert:
Runtime Insert sorted which in worst case must iterate all elements which is O(n)
Memory Constant Node struct allocation which is constant O(1)

SLRemove:
Runtime: Find the obj in list before removing. So the worst case which would be is if it is not there then it would take linear time O(n)
Memory: frees memory in constant time O(1)

SLDestroy:
Runtime: Iterates through each node and frees it which is n operations. And frees the list pointer as well which becomes n + 1 frees. Linear O(n)
Memory: Frees memory in consant time O(1)

SLCreateIterator:
Runtime: Single alloc call Constant O(1)
Memory: Size of SortedListIterator memory allocation ConstantO(1)

SldestroyIterator:
Runtime: single call and frees Constant O(1)
Memory: No variables are being created Constant O(1)

Adjust
Runtime: adjusts pointer of the iterator if any changes have been made. In worst case, it iterates throughout all n elements which is linear time. O(n)

SlNextItem
Runtime: assignments in Constant time O(1)
Memory: run and return assignments Constant O(1)

SLGetItem
Runtime: retreives an item, worst case if it is not there, therefore, linear O(n)
Memory: no memory allocation