Big-O Analysys:

Run-Time of Each Method:
SLCreate - O(1)
CreateNode - O(1)
SLInsert - O(n)
SLRemove - O(n)
SLCreateIterator - O(1)
SLGetItem - O(1)
SLNextItem - O(1)
SLDestroy - O(1)
SLDestroyIterator - O(1)

SLInsert runs in O(n). If the item being inserted is the smallest value in the list, it must traverse through n items to be inserted. Each new item is compared to the list before insertion. Therefore SLInsert runs in O(n). The best case for SLInsert would be if it is the largest item and is inserted at the head of the list. SLRemove runs in O(n). If the item to be removed is the smallest and last item, then like SLInsert it must traverse through n items to be removed. The best case for SLRemove would be if the item to be removed was the first item in the list. SLCreate, SLCreateIterator, SLGetItem, and SLNextItem all run in O(1) time. CreateNode() is a function that we created for easier node creation. It also runs in O(1). Since our program will run n times, the overall run time ends up being $O(n^2)$.

Memory Usage Analysis:

We had no memory leaks since we lost 0 bytes . We called malloc() whenever a new node was created to insert in SLInsert(), in SLCreate() for the SortedListPtr and in SLCreateIterator() for the SortedListIteratorPtr. The memory usage is O(n).