# SORTED-LIST README

Justin Chong & Matthew Bauer
Assignment 2
Systems Programming (01:198:214)
Rutgers University

## SLCreate

Runs in the best and worst case O(1) time, because regardless of the input, it will do a fixed number of operations.
It's memory usage is best and worst case O(1) time as well because the function will always allocate a fixed amount of memory.

## SLDestroy

In the best case will run in O(1) time because it will only have to free the empty head node and the SortedList struct.
In the worst case, it will run in O(n) time, where n is the input size, because in the worst case it will need to free every node in the list.
As for memory usage, the best and worst case are O(1) because no additional memory is allocated when the function is called.

## SLInsert

In the best case will run in O(1) time because in this scenario it will insert the Node containing the data at the beginning of the list,
and so the operations needed will be fixed. As for the worst case runtime, it is O(n) because in the worst case the Node containing the data must be inserted at the end of the list. Thus, it would have to traverse and check all the nodes in the list which is n nodes. The memory usage is best and worse case O(1), because regardless of the size of the list there will be a fixed amount of allocation. In the best case the data is already in the list and nothing needs to be allocated, or, in the worst case the data does need to be inserted in the list and a Node is allocated.

## SLRemove

In the best case will find the data that needs to be removed in the first Node containing data, and will run in O(1) time. If the worst case occurs, then it will find that the data is not in the list, which means it ran through all n objects in the list, and will run in O(n) time. For memory allocation, the function does not allocate any additional memory, and so in the best and worst case, has O(1) memory usage.

## SLDestroyIterator

In the best and worst case run in O(1) time because it has a fixed number of operations to perform that does not depend on input size.
The worst and best case memory usage is O(1) because it does not allocate any extra memory.

## SLNextItem

In the best and worst case run in O(1) time because acquiring the next item only needs a set number of operations and the list size is irrelevant. The worst and best case memory usage is O(1) because it only moves a pointer and doesn't allocate any memory.

## SLGetItem

In the best and worst case run in O(1) time because no matter the case all it will do is return the data value, or a NULL value if there isn't data to return. The function in the best and worst case is O(1) for memory usage because it only returns the data that is within the node that is pointed to and so it does not need to allocate any memory for any reason.