Bryan Bergo, CS2400 10/17/23

| | ResizeableArraySet | | | LinkedSet | | |
|---|---|---|---|---|---|---|
| | add(**new Entry**) | remove() | remove(**anEntry**) | add(**new Entry**) | remove() | remove(**anEntry**) |
| Best Case Time Complexity | O(1): Creating a new set and inserting a **newEntry** into it | O(1): Removing the last entry from a Resizeable ArraySet is always O(1) since the index of the last element can always be computed using the number of entries. | O(1): **anEntry** happens to be the first element inserted into a ResizeableArraySet, so no traversal is required other than the first iteration of the search loop. | O(1): **anEntry** happens to be the first element in a LinkedSet, and therefore there is no traversal necessary. | O(1): the LinkedSet happens to be only one element long, meaning no traversal is necessary. | O(1): **anEntry** happens to be the first element in a LinkedSet, so no traversal is necessary |
| Worst Case Time Complexity | O(n): in order to add **newEntry** to a ResizeableArray Set, we must traverse the Set to ensure it doesn't already exist, which takes O(n) time where n is the number of items within a set before **newEntry** is added. | | O(n): the index of **anEntry** must be found iteratively, and if it is at the end of an array, it would take n iterations over the set to find **anEntry**, where n is the number of items in a ResizeableArraySet. | O(n): LinkedSet must add items to the end of the list, meaning it will take n traversals to insert **anEntry**, where n is the number of items in the LinkedSet before **anEntry** is inserted. | O(n): in order to find and remove the last element from a set, we must traverse it to reach the end (assuming we are not actively keeping track of the tail). | O(n): **anEntry** happens to be the last element in a LinkedSet, meaning it will take n traversals to reach the end of a LinkedSet, where n is the number of entries in a LinkedSet before **anEntry** is removed |