Wood, Michael

Lab 7 big o analysis

* addToFront on the java.util.ArrayList implementation
* addToFront on the java.util.LinkedList implementation
* indexedContains on the datastructures.ArrayList implementation
* indexedContains on the datastructures.LinkedList implementation
* indexedContains on the datastructures.LinkedListTurbo implementation

1. addToFront on java.util.ArrayList is O(n) because it adds the element to the front of the new array that it size() + 1 in size. It then shifts n amount of elements down 1 to the right in the array.
2. addToFront on java.util.LinkedList is O(1) because it only needs to update where head points to and it doesn’t need to shift any elements down.
3. indexedContains on datastructures.ArrayList is O(n) because indexedContains has to loop through n amount of items in the list on worst case scenario. get on datastructures.ArrayList is only O(1) so that won’t add any additional time which is why it is only O(n)
4. indexedContains on datastructures.LinkedList is O(n2) because for contains it must walk through the entire list on worst case with is n but then at each index it calls get at that index which also walks through the entire list on worst case scenario with is another n making it O(n2)
5. indexedContains on datastructures.LinkedListTurbo is O(n2) because for contains it must walk through the entire list on worst case which is n and then the get method at worst case must also walk through the entire list which is another n making it O(n2)