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(n) because for contains it must walk through the entire list on worst case which is n and then the get method each time will only walk 1 position and it will save its last position making it O(1) making the method overall O(n)

A graph with a red line

Description automatically generated

For my Big-O Analysis I said that addToFront on java.util.ArrayList was O(n) and according to my benchmarking results you can definitely see that this is O(n) since it is a straight line.

A graph with a line

Description automatically generated

For my Big-O Analysis for addToFront on java.util.LinkedList I said that it was O(1) and at first it appears to almost be O(n) but towards the bigger list sizes it definitely appears to be O(1) likely because of optimizations for larger list sizes in the java util library.

A graph with a line

Description automatically generated

For my Big-O Analysis for indexedContains on datastructures.ArrayList I had originally said that it was O(n) and for smaller sizes it does appear to be O(n) but for larger sizes it was a lot quicker but still appears to be O(n) overall.

A graph with a red line

Description automatically generated

For my Big-O Analysis for indexedContains on datastructures.LinkedList I had originally said that it was O(n2) and the benchmarking results definitely agree with this because the times appear to be squaring for the increased list sizes.

A graph with a line

Description automatically generated

For my Big-O Analysis for indexedContains on datastructures.LinkedListTurbo I had originally said that it was O(n) but now after looking at the benchmarking results it appears to be O(logn) because the shape of the graph appears to be in a logn shape.