W08 – Lists

Design

Iterative Methods

When making the various methods I took into account that since objects are passed by reference, I would have to be cautious in how I designed them.

GetFromFront

Implementation

Iterative Methods

Size

I had to make a separate currentNode because if I used head, to iterate through the list, the value of head would change wherever the size method was initially called.

Worst Time Complexity: O(n) Linear – in order to find the size each element has top be iterated through and counted through once, which means it takes. The time complexity for this method is the same regardless of input.

Worst Size Complexity: O(n) Linear –

Contains

Worst Time Complexity: O(n) Linear – As the linked list is unordered each element has to be examined in turn. If the element we're looking for is at the end or not in the list then the entire list needs to be iterated through.

Worst Size Complexity: O(2) Constant – Only the node containing the element being compared and the element being searched for need to be stored.

Count

Worst Time Complexity: O(n) Linear – Every element in the list has to be examined to check if it matches the provided element. That means regardless of element being searched for the entire list has to be examined.

Worst Size Complexity: O(3) Constant – The only things being stored are the element being searched for, the current node and the count of matching nodes.

ConvertToString

Worst Time Complexity: O(n) Linear – The time complexity will always be linear as every element has to be iterated through in order to create a string representation of every object in the list.

Worst Size Complexity: O(2) Constant – Only the string representation and current node need to be stored.

GetFromFront

To check if an index is outside the current ListNode, I could have used the size method to check at the beginning, however this would give a linear time even in in cases where the index is zero, because the size method has linear time. In my implementation, the InvalidIndexException is only thrown if the end of the list reached.

Worst Time Complexity: O(n) Linear – If index is more longer than the length of the ListNode or at the end then every element of the ListNode has to be iterated through before an exception is thrown.

Worst Size Complexity: O(2) Constant – Only the current node and index of the element being searched for needs to be stored.

GetFromBack

I chose to use the position from the back to calculate the equivalent position from the front and use the GetFromFront method.

Worst Time Complexity: O(n) Linear – the size method is always called so the minimum time is linear. GetFromFront also has a linear time complexity, so at worst the entire list has to be iterated through twice.

Worst Size Complexity: O(1) Constant – Same as GetFromFront but a few additional things need to be stored.

DeepEquals

Worst Time Complexity: O(n) Linear – Each list is iterated through one element at a time, at the same time.

Worst Size Complexity: O(n) Linear -

DeepCopy

Worst Time Complexity: O(n) Linear – The list is iterated through once in order to copy each element.

Worst Size Complexity:

ContainsDuplicates

Worst Time Complexity: O(n^2) Quadratic – at most there can be n(n-1)/2 comparisons being made by the method. This simplifies down to n^2. This occurs if there isn't a duplicate or if only the last 2 elements in the linked list are duplicates.

Worst Size Complexity:

Append

Worst Time Complexity: O(n) Linear – The entire linked list has to be iterated through in order to append one list onto the other.

Worst Size Complexity:

Flatten

Worst Time Complexity: