Task 2: Analysis of LinkedList methods

`size`

This always will be the number of elements in the list and always runs at O(N).

`add(i, d)`

Our best case is adding to the beginning of the list because we don't have to iterate through, the worst case adding to the end of the list, as this requires iteration through all nodes. This operation runs at O(N).

`contains`

The best case is that the object is first and the worst case is that the object is the last node or not in the list, meaning we iterate through all nodes. This operation runs at O(N).

`removeFirst`

This always adds first and is not dependant on the size of the list, and therefore is constant. This operation runs at O(C).

`removeLast`

This operation requires iterating through all nodes to obtain the second to last node. This means it will always iterate size - 1 times, and therefore runs at O(N).

Task 3: Analysis of Code Segments

`Fragment A`

We iterate list.length times to count i, and then list.length times for each i to obtain j. So the total number of iteration is list.length * list.length. This is runs at $O(N^2)$ in the worst case.

`Fragment B`

We iterate list.length times to get the total, then iterate list.length more times to print out the information, this gives 2*list.length iterations. Because we ignore constants, O(2N) is reduced to O(N) in the worst case.

`Fragment C`

We iterate list.length/2 times for to count i, then iterate list.length times for each i, to obtain j. This gives list.length /2 * list.length. Because we ignore constants O(N/2*N) becomes O(N*N) or $O(N^2)$, in the worst case.