Warm-Up #1: append

Let x and L be `LinkedList Nodes`.

Analyzing append

```
1 append(x, L) {
2    Node curr = L;
3    while (curr != null && curr.next != null) {
4       curr = curr.next;
5    }
6    curr.next = x;
7 }
```

What is the...

- best case time complexity of append?
- worst case time complexity of append?

Merge

Pre-Condition: L_1 and L_2 are sorted.

Post-Condition: Return value is sorted.

Merge

```
1  merge(L1, L2) {
2     p1, p2 = 0;
3     While both lists have more elements:
4        Append the smaller element to L.
5        Increment p1 or p2, depending on which had the smaller element
6     Append any remaining elements from L1 or L2 to L
7     return L
8  }
```

What is the...

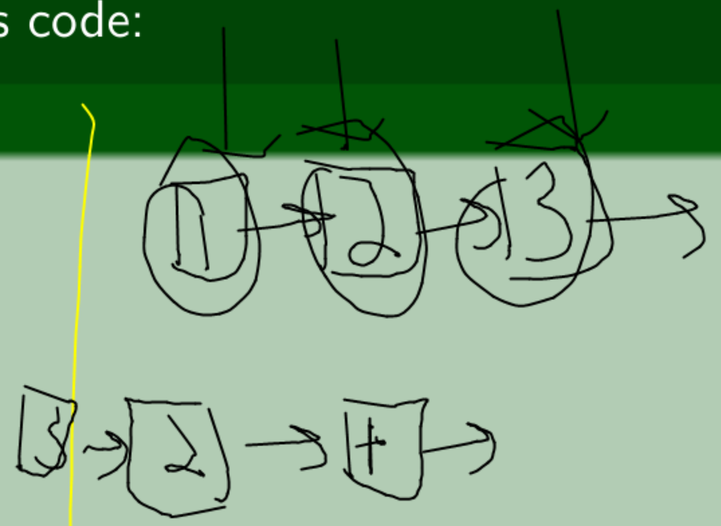
- best case # of comparisons of `merge`?
- worst case # of comparisons of `merge`?
- worst case space usage of `merge`?

Merge Sort is hard; so...

Let's start with trying to analyze this code:

LinkedList Reversal

```
1  reverse(L) {
2     if (L == null) {
3        return acc;
4     }
5     else {
6        Node front = L;
7        Node rest = L.next;
8        L.next = null;
9
10       Node restReversed = reverse(rest);
11       append(front, restReversed);
12    }
13 }
```



Notice that append is the same function from the beginning of lecture that had runtime $\mathcal{O}(n)$.

So, what is the time complexity of `reverse`?

Solving the `reverse` Recurrence

$$T(n) = \begin{cases} d_0 & \text{if } n = 0 \\ c_0 + c_1 n + T(n-1) & \text{otherwise} \end{cases}$$

$$T(n) = (c_0 + c_1 n) \quad + \quad T(n-1)$$

$$= (c_0 + c_1 n) + (c_0 + c_1 (n-1) + T(n-2))$$

$$= c_0 + c_1 n + (c_0 + c_1 (n-1)) + \cdots$$

$$+ c_0 + c_1$$

$$+ d_0$$

Analyzing Merge Sort

Merge Sort

```
1  sort(L) {
2     if (L.size() < 2) {
3        return L;
4     }
5     else {
6        int mid = L.size() / 2;
7        return merge(
8           sort(L.subList(0, mid)),
9           sort(L.subList(mid, L.size()))
10       );
11    }
12 }
```

First, we need to find the recurrence:

$$T(n) = \begin{cases} d & n = 0 \\ d & n = 1 \\ O(n) + 2T(n/2) & \text{otherwise} \end{cases}$$

Find A Big-Oh Bound For The Worst Case Runtime

```
1 sum(n) {
2     if (n < 2) {
3         return n;
4     }
5     return 2 * sum(n - 2);
6 }
```