Let $x$ and $L$ be LinkedList Nodes.

### Analyzing append

```plaintext
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. }
```

$O(n)$

### LinkedList Reversal

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

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

\[
T(n) = (c_0 + c_1 n) + T(n-1) = (c_0 + c_1 n) + (c_0 + c_1 (n-1)) + T(n-2) = \ldots (c_0 + c_1 (n-2)) + \ldots + (c_0 + c_1 (1)) + d_0
\]

\[
\sum_{i=1}^{n} (c_0 + c_i i) + d_0
\]

\[
n \cdot c_0 + c_1 \sum_{i=1}^{n} i + d_0
\]

\[
n \cdot c_0 + c_1 \left( \frac{n(n+1)}{2} \right) + d_0 = O(n^2)
\]
```c
int sum(int[] arr) {
    return help(arr, 0, arr.length);
}

int help(int[] arr, int lo, int hi) {
    if (lo == hi) return 0;
    if (lo == hi - 1) return arr[lo];
    int mid = (hi + lo) / 2;
    return help(arr, lo, mid) + help(arr, mid, hi);
}
```

The function `sum` takes an array `arr` as input and returns the sum of its elements using the `help` function. The `help` function is a recursive function that takes an array `arr` and two indices `lo` and `hi` as parameters, returning the sum of elements in the subarray from `lo` to `hi`. It uses recursion to divide the subarray into smaller parts and sums them up. The base cases are when `lo` equals `hi` or `lo` equals `hi - 1`, returning the element at the index `lo` or the sum of the current two elements, respectively. Otherwise, it calls itself recursively with the mid-index to split the subarray and sum the results.

The diagram shows the recursive calls and the function's work breakdown. The `T(N)` function is defined as:

\[ T(N) = C_3 N + C_2 N \]

where \( k = \log_2 N \) and \( k = 0 \ldots \log_2 N - 1 \). The total work is:

\[ \sum_{k=0}^{k+1} 2^k \cdot C_3 = O(N) \]