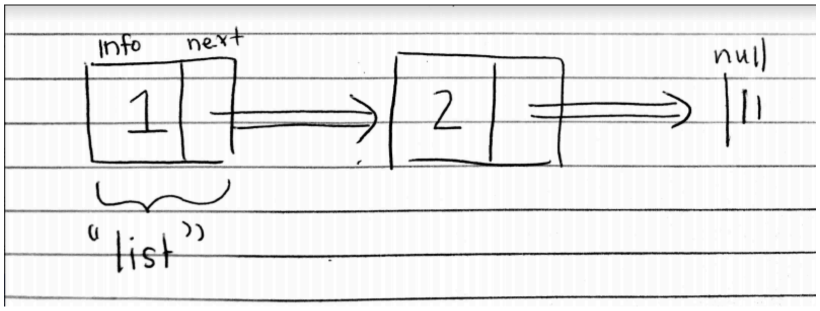
Linked Lists & Recursion

Today, you were provided with the following recursive method:

```
  public ListNode bigify(ListNode list) {
1     if (list == null) return null;
2     list.next = new ListNode(list.info, XXX );
3     return list;
   }
```

Recall that XXX can be replaced by **bigify(list.next).** I'll work through a small example to illustrate why this is true--

Here is a linked list with two ListNode elements. The parameter variable "list" is a reference to the head (i.e. first element) of the linked list.



First layer

Say you call bigify(list). Is list==null? No, it is a ListNode with value 1. Therefore, we execute line 2

- i.e. list.next = new ListNode(list.info, bigify(list.next));
    - i.e. the next element of the linked list = a new ListNode with a value of 1 and a "next" pointer to bigify(list.next)

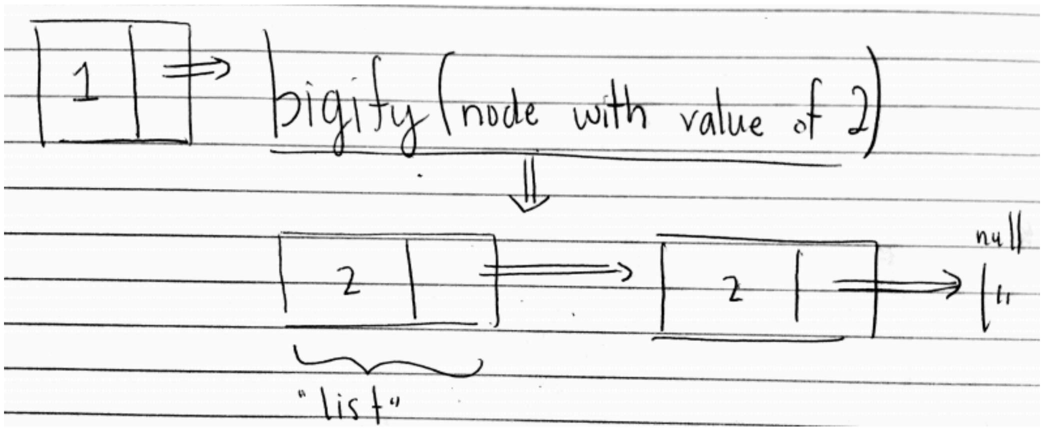
To create this new ListNode, then, we must first complete bigify(list.next), i.e. execute bigify on the ListNode with value 2.

Second layer

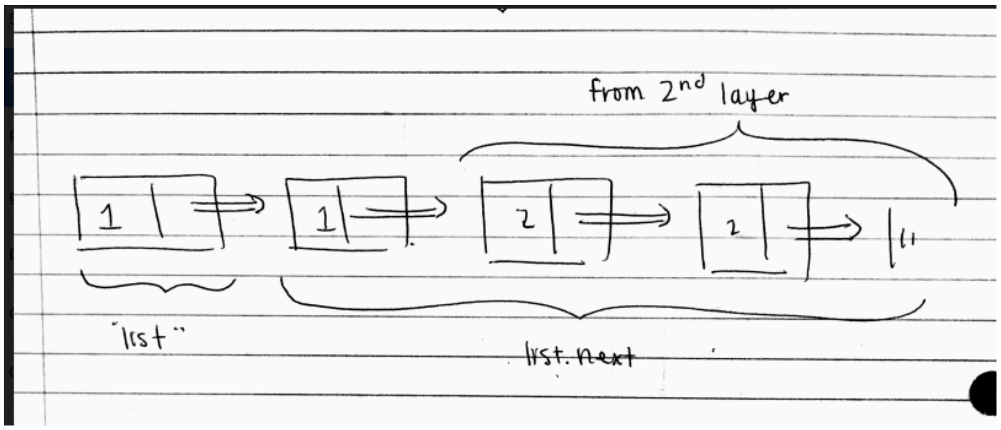
Now, the parameter variable "list" refers to what was previously list.next, the ListNode with value 2. Is list==null? No, so execute line 2 again:

- list.next = new ListNode(list.info, bigify(list.next));
    - i.e. the next element of the linked list = a new ListNode with a value of 2 and a next pointer to bigify(list.next)

To create this new ListNode, we must complete another recursive call, bigify(list.next). We know that list.next is null (since the ListNode with value 2 is the last element in the linked list, and its "next" pointer is set to null). Given line 1 of the code, we know that calling bigify(list.next) will return null.
Therefore, we can confidently assign the value-of-2 ListNode to have a "next" pointer to a ListNode with a value of 2. This new next ListNode will point to null.



Now, we collapse back to the first layer. At the first layer, we now know what bigify(list.next) [i.e. bigify called on ListNode with value of 2] is -- that is the linked list of two ListNodes (both with value of 2) that was returned in our second layer. Knowing this means that we can now assign list.next to a ListNode with a value of 1, with a next pointer to the linked list of two ListNodes with value 2.



Finally, you return list, which is the first ListNode (that has a value of 1), the original head of the linked list.

General tips for solving recursion problems

1. Consider the base case -- how will we know when we're done solving?
2. Consider how we move closer and closer to that base case so we don't just truck deeper and deeper into recursion
3. Figure out how you can incrementally solve the problem before diving deeper into recursion

How did we do that in this example?
1. We know that we won't have anything to bigify if the list is null. Therefore, our base case is "if (list == null)"
2. We know that we can move closer to null by traversing the list. Therefore, we can safely assume that our recursion will involve using list.next as an argument into the recursively-used method
3. We can solve our problem each time by creating a duplicate of the ListNode that we are currently at before determining what linked list elements we will have to duplicate later. That is the basis of the new ListNode creation in line 2.

More linked list practice

Here is a video with a more detailed overview of linked lists

If you would like more practice with linked lists, I recommend LeetCode, an online resource often used for technical interview preparation. You can type your code directly in the embedded editor and test it using their provided test cases, much like the APT tester. Some of the problems have good solutions, and all of the problems have a "Discussion" tab where people have posted their solutions. Click here to get beginner-level linked list questions. Note that, though these problems are marked as easy, they may be more challenging than the ones you've been assigned, so don't fret if you find yourself in need of help.