# Recitation 8 Recursion & Linked Lists

Ben Stoddard Tabitha Peck

#### **Covered in Recitation**

More Recursion!

**Linked Lists!** 

**Using Recursion for Linked Lists!** 

# Why Recursion?

Many of these methods would work better with loops.

Recursion will be very useful to you later (trees, midterms, finals) though, so it's better to practice now.

These problems are another chance to think about how recursion works. What are your base cases? How do you simplify the problem?

#### Unique to this Recitation

- Private Variables
  - Using Getter/Setter methods
- Private Helper Methods
  - User calls the public method for the answer, and the public method uses the private helper methods
    - In this recitation, the helper methods will be recursive (see next slide)

## Add To Tail (Example)

```
Non-Recursive
       public void addToTailNoRecursion(int value) {
         if(head == null) { ← Base Case: No Head (Empty List)
           head = new IntListNode(value);
         } else {
            IntListNode current = head;
            while(current.getNext() != null) {
              current = current.getNext(); ← Loop to advance
Add New → current.setNext(new IntListNode(value));
         size++;
```

#### Recursive

```
public void addToTail(int value){
  if(head == null) { ← Base Case: No Head (Empty List)
    head = new IntListNode(value);
  } else {
    addToTail(value, head); ← Call to Private Helper
    size++;
     Private Helper:
private void addToTail(int value, IntListNode n) {
  if(n.getNext() == null) { ← Base Case: No Next Node
    n.setNext(new IntListNode(value));
  } else {
                                         ← Recursive Case:
                                         Call again with next
         addToTail(value, n.getNext());
                                         node (advances the
                                         "pointer").
```

## What You Will Be Doing:

Fill out the rest of the private helper function (which are recursive).

- Contains: Check for a value.
- countOccurences: Counts a value.
- sum: Sums the list.
- sumEven: Sums only the even values in the list.
- reverseList: Reurns the same list, but in reverse order.
- mergeLists: Merges this list with another and returns the resulting list.
  - \*Assumes both lists are sorted. Result should be sorted as well.

Do as much as you can, and do it all recursively. Problems should be listed in increasing order of difficulty.

#### Go For It!

You shouldn't have to change any given code.

You should be able to pass unit tests if you are done.

These slides are online for your reference.

Please do as much as you can (asking UTAs for help if you get stuck), and submit what you finish **before** you leave.