## Objective

- Implement a Min-Heap using a `ListBinaryTree` as the underlying data structure.

## Setup

1. Download the `Lab30StarterCode.zip` file from `msBrekke.com`
2. Unzip starter code in `H:/CSIII/Lab30` folder

## Part 1: ListBinaryTree

Copy your completed `ListBinaryTree.java` file from your `Lab29` folder into your `Lab30` folder.

**Add** a new method to `ListBinaryTree.java`. You will use this method in your `Heap.java` file.

```
protected E removeLast()
{
  return list.remove(size()-1);
}
```

## Part 2: Adding to the Heap

Implement the `add` method in `Heap.java`. This method should add data to tree exactly like a `ListBinaryTree` (use `super`!) and then it should use the `addHelper` method to `swap` data into the correct position.

The `addHelper` method is a recursive method that should `swap` the value at `index` with its parent as long as the value at `index` is less than the parent index (because you are implementing a MIN heap). It should keep swapping (think recursively) until the heap property is met.

## Part 3: Remove Root

You will write the `removeRoot` method, but first you will write a couple helper methods.

### meetsHeapProperty

Write the helper method `meetsHeapProperty`. This method should return `true` if the value at `index` is less than the values of both of its children. If it has no children, then it should return `true`. If either child has a value less than the value at `index`, then this method should return `false`.

### getSmallestChildIndex

Write the helper method `getSmallestChildIndex`. This method should return the index of the child that has the smallest value. If the specified node has no children, then this method should return -1.

### removeRoot

Write the method `removeRoot`. This method will remove and return the value at the root of the heap. If the heap is empty, then it should return `null`. Otherwise, it should store the root value in a temporary variable, `swap` the root value with the last value in the heap, and then remove the last value from the heap (remember you added a helper method to `ListBinaryTree` called `removeLast`)

Then, *while* the value that was just swapped into the root does not meet the heap property (use the helper method you just wrote!) that value should be swapped with its smallest child.

## Part 4: Heapify

Write the `heapify` and `sink` methods. The `heapify` method should call the sink method on each index of the heap, **backwards**.

The `sink` method should swap the value at `index` with its smallest child if the value at `index` does **not** meet the heap property. It should recursively do this until the heap property is met. (use your helper methods!)

## Grading

Use the *Lab30Tester* tests to make sure your code works properly. This lab will be tested auto-magically.

| | |
|---|---|
| Part 1 | 5 points |
| Part 2 | 20 points |
| Part 3 | 50 points |
| Part 4 | 25 points |