Induction

Sometimes it can be difficult to convince yourself and others that a function generates the correct results for all possible input. For example, consider the following function which purports to compute the square of its arguments:

```
square :: Int -> Int
square n = if n==0
               then 0
               else square (n - 1) + (2 * n - 1)
```

How would we show that this function correctly squares all nonnegative integers? It correctly squares 0. *If* we can assume that $square(n-1)$ correctly gives us the result of $(n-1)^2$ then with a bit of algebra we can show that $(n-1)^2+2n-1=n^2$. In other words, to show that *square* n words we had to assume that $square(n-1)$ works. That feels like circular reasoning, but we really aren't cheating here. We assumed that *square* worked correctly on a *smaller* value. To show that the smaller values works we could then assume that it works on a value even smaller. This leads to a chain of reasoning which will eventually end when we get to the case of *square* 0, which we already know works.

By contrast, we can show by algebra that $n^2 = (n+1)^2 - (2n+1)$. Even so, the following function does not work correctly:

```
square :: Int -> Int
square n = if n==0
               then 0
               else square (n + 1) - (2 * n + 1)
```

Why doesn't this new version of *square* work? It will end in infinite recursion since it computes values further and further from the base case.

The reasoning we used to argue correctness of our working version of square is called *mathematical induction*. Such reasoning can be broken into three parts:

1. We require a base case of the proof to end the chain.
2. We assume that the function works for a smaller value. This is the induction hypothesis and we need this to prove the *if* in the reasoning chain.
3. Finally the reasoning that leads from the inductive hypothesis to the conclusion is the inductive step.

Here is the proof for the correctness of our square function:

1. The base case is when n is 0. `square 0` returns the value of 0 which of course is 0^2, so `square 0` returns the correct result.
2. We assume that the `square k` returns the correct result for values less than n. In other words, `square k` returns k^2 whenever $0 \leq k < n$.

3. Finally we need to show that `square n` also returns the correct result. We go back to the definition that `square n = square (n-1) + (2*n-1)`. Because $n-1 < n$, we can use the previous step to argue that `square (n-1)` returns $(n-1)^2$. Therefore the return value of `square n` is $(n-1)^2 + (2n-1) = n^2 - 2n + 1 + 2n - 1 = n^2$.

The three steps together gives us the proof that this version of `square` works correctly for all nonnegative inputs. This type of proof is also useful when you are trying to debug a program that does not work correctly. If we tried to prove our incorrect square function using induction, we would run into trouble with the inductive step. Why? This should point you toward the bug in the function.

Here is another version of the function `square` with a bug in it:

```
square :: Int -> Int
square n = if n==0
               then 0
               else square (n - 2) + (4 * n - 4)
```

Try to prove it is correct with induction. Where does the proof run into trouble? Where is the bug?