Warmup

1. Write a recursive function \( \text{log}(b : \text{Int}, n : \text{Int}) : \text{Int} \) that computes \( \lfloor \log_b n \rfloor \).
   \[ \text{Solution:} \quad \text{def log}(b : \text{Int}, n : \text{Int}) : \text{Int} = \text{if } (b < n) \text{ then } 0 \text{ else } 1 + \text{log}(b, n / b) \]

2. Is our solution tail recursive? If not, can we write one that is tail recursive?
   \[ \text{Solution: No. Yes:} \]
   \[ \text{def log}(b : \text{Int}, n : \text{Int}) : \text{Int} = \{
   \quad \text{require(b > 0 \&\& n > 0)}
   \quad \text{def logHelper}(b : \text{Int}, n : \text{Int}, acc : \text{Int}) : \text{Int} = \{
   \quad \quad \text{if } (b < n) \text{ then } acc \text{ else } logHelper(b, n / b, acc + 1)
   \quad \}
   \quad \text{logHelper}(b, n, 0)
   \} \]

3. (a) What is the advantage of the tail recursive solution?
   (b) What advantages can a non-tail recursive solution have?
   \[ \text{Solution: (a) A smart compiler/interpreter (such as the Scala compiler/Lisp interpreter) can optimize away tail recursion into a loop. Note that many compilers/interpreters do not or cannot do this (such as the Java compiler or Python interpreter). To test for yourself, write a tail recursive function and see if you cause a stack overflow by passing your function a large value (> 1000 or so)!
   (b) Often, non-tail recursive solutions are cleaner and closer to what we are trying to express (our non-tail recursive \textit{factorial}() is a much better example of this than our non-tail recursive \textit{log}() function) \]

4. Write a recursive function \( \text{sum}(n : \text{Int}) : \text{Int} \) that returns the sum of the integers from 1 to \( n \).
   \[ \text{Solution:} \quad \text{def sum}(n : \text{Int}) : \text{Int} = \text{if } (n == 1) \text{ then } 1 \text{ else } n + \text{sum}(n - 1) \]

5. Prove by mathematical induction that your \textit{sum()} function returns the sum values from 1 to \( n \) (hint: a closed form formula for the sum of the values from 1 to \( n \) is \( n \times (n + 1)/2 \))
   \[ \text{Solution: Theorem:} \quad \text{sum}(n) = \sum_{i=1}^{n} i. \]
   \[ \text{Proof: By mathematical induction on} \ n. \]
   \[ \text{Base Case:} \ n == 1. \quad \text{sum}(1) = 1 = 1 \]
Inductive Hypothesis: $\sum(n') \rightarrow^* \sum_{i=1}^{n'} i$, where $n' = n - 1$.

We will show that $\sum(n) \rightarrow^* \sum_{i=1}^{n} i$:

$\sum(n) \rightarrow^* n + \sum_{i=1}^{n'} i$. (by definition of $\sum(n)$ and IH)

$= n + (n' * (n'+1) / 2)$ (using formula for summation)

$= n + ((n - 1) * (n-1)+1) / 2$ (substituting for $n'$)

$= n + ((n - 1) * n / 2$ (arithmetic)

$= 2n/2 + ((n - 1) * n / 2$ (arithmetic)

$= (2n + (n - 1) * n) / 2$ (arithmetic)

$= (n^2 + n) / 2$ (arithmetic)

$= n * (n+1) / 2$ (arithmetic)

$= \sum_{i=1}^{n} i$ (using formula for summation). □

Recursion

1. Write a recursive function $\text{isPrime}(n : \text{Int}) : \text{Boolean}$ that returns true if and only if $n$ is a prime number (hint: use a helper function).

Solution:

```scala
def isPrime(n : Int) : Boolean = {
  def isPrimeHelper(n : Int, range : Int) : Boolean = {
    if (range > n / 2) true
    else if (n % range == 0) false
    else isPrimeHelper(n, range + 1)
  }
  n > 1 && isPrimeHelper(n, 2)
}
```

Note: Mathematically, this solution is incredibly naïve! Can you think of some ways to improve it?

Induction

1. Given the height $h$ of a stack of cannonballs, we can compute the number of cannonballs in the stack using the formula $C(h) = h^2 + (h - 1)^2 + \ldots + 1^2$.

(a) Write a recursive function $\text{countCannonballs}(h : \text{Int}) : \text{Int}$ that computes the number of cannonballs in a stack given the height of the stack. You can assume that all stacks will have at least one cannonball.

(b) Prove by mathematical induction that your function computes the same result as the cannonball relation.
For this stack, \( h = 5 \), so \( C(5) = 5^2 + 4^2 + 3^2 + 2^2 + 1^2 = 55 \) cannonballs.

Solution:
(a) \( \text{def countCannonballs (h : Int) : Int = if (h == 1) 1 else (h * h) + countCannonballs(h - 1)} \)

(b) Theorem: \( \text{countCannonballs}(h) \rightarrow^{*} C(h) \).
Proof: By mathematical induction on \( h \).
Base Case: \( h == 1. \) \( \text{countCannonballs}(1) \rightarrow 1 == 1 \)
Inductive Hypothesis: \( \text{countCannonballs}(h') \rightarrow^{*} C(h') \), where \( h' = h - 1 \).
We will show that \( \text{countCannonballs}(h) \rightarrow^{*} C(h) \):
\[
= h * h * \text{countCannonballs}(h - 1) \quad \text{(by definition of countCannonballs())}
= h * h * C(h') \quad \text{(by substitution for h and IH)}
= h * h * h'^2 * (h'-1)^2 * ... * 1^2 \quad \text{(expanding C(h'))}
= h * h * (h-1)^2 * (h-2)^2 * ... * 1^2 \quad \text{(substituting for h' and arithmetic)}
= h^2 * (h-1)^2 * (h-2)^2 * ... * 1^2 \quad \text{(arithmetic)}
= C(h) \quad \text{(definition of C(h))}. \]