COMS10014 Worksheet 6: Induction

1. Induction over the Integers

(★★) Use proof by induction to prove the following properties $P(n)$, either for $n = 0, 1, 2, \dots$ or for $n = 1, 2, \dots$ as appropriate.

1. $$P(n)\colon \sum_{i=1}^{n} (2i-1) = n^2$$
2. $$P(n)\colon \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$
3. $$P(n)\colon \sum_{i=0}^{n} r^i = \frac{1 - r^{n+1}}{1 - r} \text{ for } r \neq 1$$
4. (★★★)

   $n^3 - n$ is divisible by 3 for all $n \in \mathbb{N}$

2. Sums of Squares (★★)

a. Identify a pattern in the following (write out more terms if you need to) and find a formula to compute n^2 from $(n-1)^2$:

n=1	$n^2 = 1$	
n=2	$n^2 = 4$	= 1 + 3
n=3	$n^2 = 9$	= 4 + 5
n=4	$n^2 = 16$	= 9 + 7
n=5	$n^2 = 25$	= 16 + 9
n=6	$n^2 = 36$	= 25 + 11

b. Prove your formula without induction.

c. Prove your formula by induction.
Note, it is only a proof by induction if you use the induction assumption in the induction step! Otherwise you are just doing a direct proof, or you may have made a mistake.

3. Prime Decomposition (★★★)

Prove that every integer $n \geq 2$ has a prime decomposition, that is we can write
$$n = p \times q \times r \times \dots$$
where all the factors $p, q, r, \dots$ are prime numbers. The same prime may appear more than once.

Note: we are not asking you to prove that prime decomposition is unique, even though that is true.

a. Use any proof strategy that is not induction.
b. Use proof by strong induction (see the note on the next page).

Note: One of the key steps is the same in both the proof approaches. For the induction proof, you need a technique called *strong induction*: instead of $P(n) \vdash P(n+1)$, the induction step is $\big(P(1) \wedge \ldots \wedge P(n)\big) \vdash P(n+1)$. This is allowed, and follows from the same principle of well-ordering on the natural numbers as the normal induction principle.

4. Structural Induction (★★★)

We consider strings composed of the three characters 💧 (water), 🔥 (fire) and 🍃 (air – represented by a blowing leaf) according to the following rules:

1. Starting string: the string 🔥🍃 (fire, air) is a valid string.
2. Rain rule: on the end of any string ending with 🍃(air), you may add 💧 (water) to form another valid string.
3. Sun rule: For any string starting with 🔥 (fire) and containing at least one other symbol, you may form a new valid string starting with 🔥 (fire) and containing the rest of the original string twice in a row.
4. Wind rule: if a valid string contains three 🍃(air) symbols in a row, you may replace them with one 💧 (water) symbol to form another valid string.

For example, here is a valid derivation using these rules:

	🔥🍃	starting string
$\vdash$	🔥🍃🍃	sun rule
$\vdash$	🔥🍃🍃🍃🍃	sun rule
$\vdash$	🔥💧🍃	wind rule
$\vdash$	🔥💧🍃💧	rain rule
$\vdash$	🔥💧🍃💧💧🍃💧	sun rule

Show, by structural induction, that the string 🔥💧 (fire, water) is not a valid string under these rules. To do this, find some invariant that all valid strings must have, prove the invariant with structural induction, and then show that the fire-water string does not have the invariant.

5. Invariant Induction: Square-and-Multiply (★★★)

In some forms of cryptography, we want to compute $y = g^x \pmod p$. Typically, x is a secret key (an integer less than p) and y is the matching public key; $g > 0$ is a publicly known natural number (less than p), and p is a publicly known prime number.

The most well-known application of this is so-called Diffie-Hellman key exchange, where two people who do not share a common secret can come together and create a shared secret over a public communication channel (there is a whole page of caveats here, but that's the job of the 3rd year Cryptography unit to explain).

One algorithm to do this is called square-and-multiply. It relies on the following facts about modular arithmetic:

- $g^a g^b \pmod p = g^{a+b} \pmod p$ and $(g^a)^b = g^{ab} \pmod p$
- $a \times b \pmod p = \big(a \pmod p\big) \times \big(b \pmod p\big) \pmod p$

The second rule says that if you want to compute a product $a \times b$ modulo p then you can first, if a, b are larger than p, reduce both of them modulo p before multiplying, as long as you reduce again at the end. This means you never need to work with numbers bigger than p^2, which is good since p in cryptography can be thousands of bits long.

Suppose you want to compute 11^5 modulo 13:

$$\begin{aligned}
& 11^5 & \\
\equiv\ & (11 \times 11) \times 11 \times 11 \times 11 & \\
\equiv\ & 121 \times 11 \times 11 \times 11 & \\
\equiv\ & 4 \times 11 \times 11 \times 11 & \text{since } 121 \equiv 4 \pmod{13} \\
\equiv\ & 44 \times 11 \times 11 & \\
\equiv\ & 5 \times 11 \times 11 & \text{since } 44 \equiv 5 \pmod{13} \\
\equiv\ & 55 \times 11 & \\
\equiv\ & 3 \times 11 & \text{since } 55 \equiv 3 \pmod{13} \\
\equiv\ & 33 & \\
\equiv\ & 7 & \text{since } 33 = 2 \times 13 + 7 \text{, so } 33 \equiv 7 \pmod{13}
\end{aligned}$$

To compute $y = g^x \pmod{p}$, the idea of square-and-multiply is to write out x in binary as $x_k \dots x_1 x_0$ where each x_i is a bit, with x_0 the least significant bit. The algorithm is then:

```
// input: natural numbers k, x, g > 0, p with p prime and x is given as an array
// x[0] ... x[k] (in practice one would use bit-operations here)
// output: natural number y that is the remainder of g ** x modulo p
// % is the modulo operator
y = 1
a = g
for (i = 0; i < k; i = i + 1) do
    if (x[i] == 1) then
        y = (y * a) % p     // the “multiply” part
    end if
    a = (a * a) % p         // the “square” part
end while
```

Show, using invariant induction, that this algorithm computes the correct result.
(You do not need a variant, as the “for” loop will always terminate when written like this.)

You may assume that nothing “wraps around” due to machine restrictions, that is, all integers involved are “big integers” that can store as large values as they need.

Warning: if you implement the algorithm like this to do real cryptography, it is insecure – more details in the cryptography units. Please do not use the code here to create a new cryptocurrency without understanding what the problem is.