# FIT2004: Tutorial 1 (held in Week 3)
# Covering concepts from Weeks 1 & 2

**Objectives:** The tutorials, in general, give practice in problem solving, in analysis of algorithms and data-structures, and in mathematics and logic useful in the above.

**Instructions to the class:** Prepare your answers to the questions **before** the tutorial! It will probably not be possible to cover all questions unless the class has prepared them in advance. There is 1 mark worth for this Tute towards **active** participation. 0.5 marks is towards answering the starred questions (*) indicated below, **before** attending your assigned tutorial. You will have to hand in your work on these starred questions to your tutor at the very start of the tutorial. Remaining 0.5 mark is for participating during the rest of the tutorial.

**Instructions to Tutors:**

i. The purpose of the tutorials is not to solve the practical exercises!

ii. The purpose is to check answers, and to discuss particular sticking points, not to simply make answers available.

1. Use mathematical induction to prove the following:

$$\sum_{i=0}^{N} ar^i = a + ar + ar^2 + ar^3 + \cdots + ar^N = \frac{a\left(r^{N+1} - 1\right)}{r - 1}$$

2. Use mathematical induction to prove the following:

$$\sum_{i=1}^{N} i^3 = \frac{N^2 \left(N + 1\right)^2}{4}$$

3. Listed below is the list abstract data type introduced in the lecture 1.2 (see l1.2.pdf on moodle):

```
type list e = nil | cons(e * (list e))
```

Recall (see slide 10 in l1.2.pdf), we defined an algorithm called `append` as follows:

```
append(nil, L2) = L2
| append(L1,L2) = append(cons(h,T_L1), L2)
                = cons(h,append(T_L1,L2))
```

Your task is to revise the *formal proof* given on slides 11-12 that shows that, for three lists `L1`, `L2` and `L3`, the `append` algorithm is associative:

```
append(L1,(append(L2,L3))) = append(append(L1,L2),L3)
```

4.* Listed below is the abstract data type definition for a binary-tree data structure (similar to what's been introduced in the lecture 1.2):

```
type tree e = nilTree | fork e * (tree e) * (tree e)
```

Further, below are definitions of two algorithm on this data structure:

`M` applicable to trees of any kind, defined as:

```
M(nilTree)  = nilTree
|  M( fork(e,L,R) ) = fork(e, M(R), M(L))
```

`SUM` on trees of numbers (integers or floating-point):

```
SUM(nilTree) = 0
|  SUM( fork(e,L,R) ) = e +  SUM(L) + SUM(R)
```

Your task in this tutorial is prove some properties of the algorithms:

(a) Prove **formally** that M(M(T)) = T, for every tree T.

(b) Prove **formally** that SUM(M(T)) = SUM(T), for every tree T of numbers.

5.* Show that the following recurrence relationship:

$$T_N = \begin{cases} 2 * T_{N-1} + a, & \text{if } N > 0 \\ b & \text{if } N = 0. \end{cases}$$

has the solution:

$$T_N = 2^N * (a + b) - a$$

-=END=-