FIT2004: Tutorial 2 (held in Week 5)
Covering concepts from Weeks 2, 3 & 4

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 0.5 mark worth for this Tute towards active participation. 0.25 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 0.25 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. * The Fibonacci sequence is usually defined as follows:

```
fibonacci(1) = fibonacci(2) = 1
fibonacci(n) = fibonacci(n-1) + fibonacci(n-2),  where n>2
```

It is easy to write (**a highly inefficient**) recursive program to compute the nth number in the Fibonacci sequence. For instance:

```
1 fibonacci(n)
2 {
3  if( n <= 2 ) return 1;
4  else return fibonacci(n-1) + fibonacci(n-2);
5 }
```

- What is the computational complexity of this algorithm? Attempt to prove it formally!
- Can you write a more efficient version that is NOT iterative, but instead single-recursive (rather than double-recursive as in the version above)?

- What is the time complexity of such a single-recursive implementation?

2. * Given the following algorithm:

```
for i from Lo1 to Hi1 do
   for j from Lo2 to Hi2 do
      body()
   end_for
end_for
```

   How many times is **body()** executed for the following values

```
Lo1=1, Hi1=10, Lo2=i, Hi2=10,
Lo1=0, Hi1= 9, Lo2=0, Hi2= 9,
Lo1=1, Hi1=n, Lo2=i-2, Hi2=i+2,
Lo1=0, Hi1=n, Lo2=i, Hi2=2*i ?
```

3. Given the following algorithm how many times are

   (a) **base()**

   (b) **body()**

   executed for the calls, **r(10)**, **r(5)**, and **r(1)**?

```
function r(n)
 { if( n > 0 )
    { body();
      r(n-1);
    }
   else
      base();
 }
```

   Why?

4. * Given a sequence of numbers, the longest increasing subsequence problem is to find the longest subsequence in the list which is sorted in ascending order. For example, if a list is $\{0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15\}$ then the longest increasing subsequence is $\{0, 2, 6, 9, 11, 15\}$ because there is no subsequence in this list that is in ascending order and is longer than this. Write your ideas on how this problem can be efficiently solved using dynamic programming.

5. Using arguments similar to those used in lecture 3 for average-case analysis of Quick Sort, show that the average-case complexity of Quick Select is O(N).