CS2011 Tutorial Sheet 2: Algorithms and their Complexity

This is to occupy one tutorial session. It covers material in Chapter 2 and Chapter 3 in the course textbook (*Fundamentals of Algorithmics*, by Brassard and Bratley, 1996). You should attempt the questions, writing out full answers, before the tutorial. You may need to consult the course textbook to help with some of the material.

Question 1.

Be prepared to explain to your tutor the following concepts:

1. Measures of the performance of an algorithm. Time and space complexity.
2. Worst-case, average-case and best-case complexity.

Question 2.

Describe the performance of the following algorithms using suitable measures of complexity. You should make clear what operations you are counting, what is the worst-case that you are considering, (and, perhaps, average-case and best-case, where appropriate). Consider also space complexity.

1. Multiplication of two $N \times N$ matrices. Also, the N-th power of an $N \times N$ matrix, using repeated matrix multiplication,
2. Long multiplication of integers,
3. Integer division (i.e. an integer result, plus remainder) using 'long division'.

Question 3.

The definition of the Fibonacci sequence is (in SML) as follows:

```
fun fib(0) = 0 | fib(1) = 1
  | fib(n) = fib(n-1) + fib(n-2)
```

Here is an alternative algorithm (in C):

```
int fib(int n)
  { int i, FibSeq[100];
    FibSeq[0]= 0; FibSeq[1]= 1;
    for (i= 2; i < 100; i++)
       FibSeq[i]= FibSeq[i-1] + FibSeq[i-2];
    return FibsSeq[n]; }
```

This creates an array containing the whole sequence up to the 100-th item. Yet only the two immediate predecessors are required to calculate a value in the sequence. Use this idea to recast this algorithm, removing the array and

using instead two additional variables, x and y. Write it in C. Attempt to write the same (modified) algorithm in SML. Hint: First define a function of three arguments which are the position n in the sequence and the two additional variables x and y. Then the entry in the Fibonacci sequence is obtained by calling this function with x and y set to the initial values 0 and 1.

What is the (time) complexity of the original definition as a means of computing items in the Fibonacci sequence? Hint: Produce a recurrence relation and use the fact that the numbers `fib(n)` are $O(r^n)$ where r is the golden ratio, $r = 1.61803\ldots$.

What is the (time) complexity of the above algorithm in C and of your improved version?