THE UNIVERSITY OF WESTERN ONTARIO

DEPARTMENT OF COMPUTER SCIENCE
LONDON CANADA

Analysis of Algorithms

(Computer Science 3340b)

ASSIGNMENT 1
Due date: Thursday, January 29, 2015, 11:55 PM

1. Find the formula for the summation $\sum_1^n i(i+1)$. Hint: $i(i+1) = (\ (i+1)^3 - i^3 - 1\ )/3$.

2. **A complete binary tree** is defined inductively as follows. A complete binary tree of height 0 consists of 1 node which is the root. A complete binary tree of height $h+1$ consists of two complete binary trees of height h whose roots are connected to a new root. Let T be a complete binary tree of height h. Prove that the number of leaves of the tree is 2^h and the size of the tree (number of nodes in T) is $2^{h+1} - 1$.

3. Question 2-4 (pp. 41) in the text book.

4. Read the text book for the definition of o and ω and answer question 3-2 (pp. 61) in the text book.

5. Question 4-2 (pp. 107) in the text book.

6. Suppose that the running time of a recursive program is represented by the following recurrence relation:

$$\begin{aligned} T(2) &= 1 \\ T(n) &\leq 2 * T(n/2) + n\log_2(n) \end{aligned}$$

Determine the time complexity of the program using recurrence tree method.

7. Consider the Fibonacci numbers:
$F(n) = F(n-1) + F(n-2),\ n > 1;\ F(0) = 0;\ F(1) = 1.$

a). Write a recursive function to compute $F(n)$ using the above definition directly. Implement your solution and print $F(i * 5)$, where $0 \leq i \leq 7$, as output.

b). Write a recursive function/procedure to compute $F(n)$ with time complexity $O(n)$ (more precisely, the time complexity should be $O(nA(n))$ when n is large, where $A(n)$ is the complexity of adding $F(n-1)$ and $F(n-2)$). Implement your solution and print $F(i * 10)$, where $0 \leq i \leq 30$, as output. This program must be able to compute $F(n)$ precisely for $n \leq 300$. (Hint: can you use a primitive type to store $F(300)$?) You are not allowed to use large integer, such as BigInteger in Java, from the language library. You have to write your own large integer data type or object.

c) (optional for bonus credit)
Write a recursive function/procedure to compute $F(n)$ with time complexity $O(\log(n))$ (more precisely, the time complexity should be $O(\log(n)A(n))$ when n is large, where $A(n)$ is the complexity of adding or multiplying $F(i)$ and $F(j)$). Again, you are not allowed to use large integer, such as BigInteger in Java, from the language library. You have to write your own large integer data type or object.
Hint:

$$\begin{pmatrix} F(n) \\ F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F(n-1) \\ F(n-2) \end{pmatrix}$$

d) Use the Unix time facility (/usr/bin/time) to track the time needed to compute for each run and for each algorithm. Compare the results and state your conclusion in two or three sentences.

e) Can you use your program in a) to compute $F(50)$? Briefly explain your answer. Explain why your program in b) computes $F(300)$ precisely?

Use script command to capture the screen of the execution of your program into a file.

For this question, a makefile should be written such that the command "make clean" will remove all the "*.o" files, the command "make asn1_a" will generate an executable file "asn1_a" for 7 a) that can be run by typing "asn1_a", and the command "make asn1_b" will generate an executable file "asn1_b" for 7 b) that can be run by typing "asn1_b". If you are using Java, you may not need the makefile. In that case, you should have script files, "asn1_a" and "asn1_b", so that by typing them, your java programs will run.