CS 667 : Homework 3(Due: Mar 8, 2012)

This Homework is Problems 1-5 and worth 250 points. You can replace some of these points with Problem 6 or 7, but you can only submit 250 points worth of problems.

Problem 1. (50 points)
Show that interpolation requires $O(n^2)$ time in the word model of computation. (This is along Exercise 30.1-5 on page 906 of CLRS3e. You can read the hints there. Also review HW2.)

Problem 2. (50 points)
(a) We want to compute $m = m_1 m_2 \ldots m_k$. Show that m can be computed in $O(\lg m \cdot \lg m) = O(\lg^2 m)$ bit operations.
(b) You are given x, y. How many bit operations does it take to find $n = x^y$. Express the number of bit operations using asymptotic notation in terms of n (not x nor y).

Problem 3. (50 points)
(a) **Fibonacci Revisited.** Show that F_n can be computed in $O(n^2)$ bit operations even if n-bit multiplication can only be done in $\Theta(n^2)$ bit operations.
(b) How many bit operations to find $n!$? Explain. Grading will depend on amount of bit operations performed.

Problem 4. (50 points)
Observe the notation given. This problem tests your reading of the class material. You are given an a-bit positive integer A and an b-bit positive integer B. You may assume $a \geq b$. How fast can you compute $A + B$, $A - B$, $A \cdot B$, and find Q, R such that $A = BQ + R$, $0 \leq R < B$? Express your answer in terms of a and b and justify it by pointing to the appropriate reference (book or notes) or describe the algorithm. In particular, for the last question, show (proof, algorithm) that division i.e. finding, Q, R can be done at least as fast as $O(ab)$, if not faster. (This would then imply a time bound of $O(ab)$ for the gcd of A and B respectively.)

Problem 5. (50 points)
You have a machine with 2000MB of free memory. We want you to use (potentially) all of its memory to store `wordID`s and corresponding `words`, similarly to the structure shown in class Subject 2 (cases T2$*$). An array A will be used to store `words` delimited by nulls (`\0`). A hash table T will store a `wordID`s along with a reference/pointer p to A. That way a `wordID` will be associated with a specific `word`.
The Operating System consumes 70MB and other user programs including the ones dealing with A and T another 30MB. The average length of a `word` is given as 9 ASCII characters. A `wordID` and p can only be multiples of one byte (i.e. 1B, 2B, 3B but not of bits, and thus 23bits is not a option) for efficiency. Organize the hash table and the array for maximum efficiency.
What is your choice of n, the number of words that can be accommodated? What is your choice of m, the number of slots of the hash table? How many bytes for a `wordID` and how many bytes for p? How much space in MB will your scheme consume for T and for A? Explain. (Round to nearest million multiple for n, m, T, A.)

```
n       =
m       =
wordID  =
p       =
T       =
A       =
```

Problem 6. (50 POINTS)
Read also Problem 5 for background. Design a hash-table organization program that accepts from a file with a fixed name `hw3p6conf` the following input (the values of the parameters can vary and are given based on the statement of Problem 5). Your code understands units `MB, GB, B, M`, where `1GB=1000MB, 1MB=1000000B. 1M=1000000`. In addition the encoding of the words is defined by CODE which can be ASCII (1B) or UNICODE (2B). Thus, a 7 character word would need 7B or 14B for the former or latter case.

```
RAM=2000MB
OS=70MB
PROGRAMS=30MB
AVERAGE=9
WORDID=YES
CODE=ASCII
```

Your design would output the optimal values of the six unknowns specified in Problem 5. In addition, it would output $A + T$ as a seventh output (and make sure it is no more than the size of `RAM`. A difference between this and Problem 5 is the variability allowed for `WORDID` being a `YES` (as in Problem 5) or `NO` as it was the example done in class. In the latter case you don't store a `WORDID` field into the hash table, just p. Another variability is the encoding of the words in A: `ASCII` (1B per char) or `UNICODE` (2B per char) are possible.

```
   n        =
   m        =
   wordID   =
   p        =
   T        =
   A        =
 A+T        =
```

Values for T, A are to be reported in `MB` with units properly displayed. `wordID`, `p` should be in `B` with unit information also displayed and following the restrictions of Problem 5. Values `n,m` in `M` with unit symbol also displayed.
Your code in C/C++ should be run with

```
% ./hw3p6 hw3p6conf
or
% ./hw3p6                      //  Reads default hw3p6conf
```

and for Java (name classes appropriately)

```
% java hw3p6 hw3p6conf
or
% java hw3p6                   //  Reads default hw3p6conf
```

For testing, although leading spaces will be avoided in any one of the lines of `hw3p6conf`, you may not assume this to be the case for the trailing white space after the values.

Problem 7. (100 POINTS)
Implement the Karatsuba-Ofman algorithm for non-negative integers. The input/output will be decimal (base-10) integers. Whether you plan to maintain the long integers in binary or not it's up to you.

```
// The following is pseudocode
// Whether you provide an interface in which n3 is an integer, a pointer to an
// integer or a reference it's up to you.
ReadKaratsuba(file digit1, longint d1);
MultiplyKaratuba(longint d1, longint d2, longint d3);
WriteKaratsuba(longint d1, file digit1);
```

A file contains on its first line a non-negative integer indicating the number of integer digits that will follow. The remaining lines contain the digits. Line breaks, spaces, tabs might exist but are ignored. For example

```
10
12345 56
789
```

stores integer the 10-digit integer `1234556789`. Note that the length in digits is given in the first line.

I should be able to test your code from the command line with an operation of the form

```
% ./karatsuba  file1 file2 file3
or
% java karatsuba file1 file2 file3
```

where `file1, file2` contain the two integers that will be multiplied and `file3` will store the product in the same format as that described earlier.

The first function reads a long integer from a file. The second function uses the internal representation and the algorithm presented in class for binary numbers (adapt it as fit) to speed up multiplication faster than $\Theta(n^2)$. The last function prints an internally-stored integer into a file in the standard format described in this assignment. Make sure that in the latter case you never print more that 50 character/decimal digits per line.

Note. Java has extensive `BigInteger` capabilities that might include a Karatsuba implementation. Those of you who plan to use Java, you are not allowed to use this functionality, i.e. you need to implement functions from scratch, i.e. you can not even use the `BigInteger` class. Similarly for C/C++ implementors, additional libraries (eg. gmp or the like) are not allowed.