CS 667 : Homework 4(Due: Apr 4, 2013)

Problems 1-6 are for 200pts. You may replace some of them with Problem 7 or 8 for a total of 200.

Problem 1. (40 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 might and will for a correct solution somehow use the results of two consecutively-indexed problems of Homework 3.)

Problem 2. (40 POINTS)
(a) We would like to compute in the BIT model, $m = m_1 m_2 \ldots m_k$. Show that m can be computed in $O(\lg m \cdot \lg m) = O(\lg^2 m)$ bit operations.
(b) Then find the optimal number of bit operations to find $n!$ as implied by part (a)? Explain. Grading will depend on minimal amount of bit operations performed.

Problem 3. (20 POINTS)
Show that integer division of A by B to determine the integer quotient Q and remainder R exactly (not approximation of) i.e. $A = BQ + R$, $0 \le R < B$, requires time that is $O(\lg A \lg B)$.

Problem 4. (20 POINTS)
Use the result of the previous problem to show that the GCD computation requires $O(\lg a \lg b)$ bit operations thus improving the rough $O(\lg^3 a)$ bound given in class.

Problem 5. (20 POINTS)
(a) Let $M(n)$ be the time to multiply two $n \times n$ matrices and let $S(n)$ be the time to square an $n \times n$ matrix. Show that multiplying and squaring have essentially the same difficulty: i.e. an $M(n)$ matrix multiplication algorithms implies an $O(M(n))$ squaring algorithm and and an $S(n)$ squaring algorithms implies an $O(S(n))$ matrix multiplication algorithm.
(b) n class we presented an alternative to the Strassen's method that finds the product of two $n \times n$ matrices A and B using the following sequence of multiplication operations recursively

$$
\begin{aligned}
m_1 &= (A_{12} - A_{22}) * (B_{21} + B_{22}) \\
m_2 &= (A_{11} + A_{22}) * (B_{11} + B_{22}) \\
m_3 &= (A_{11} - A_{21}) * (B_{11} + B_{12}) \\
m_4 &= (A_{11} + A_{12}) * B_{22} \\
m_5 &= A_{11} * (B_{12} - B_{22}) \\
m_6 &= A_{22} * (B_{21} - B_{11}) \\
m_7 &= (A_{21} + A_{22}) * B_{11}
\end{aligned}
$$

(where A_{ij} and B_{ij} are the $n/2 \times n/2$ submatrices of A and B), and combining the products m_i's as follows to derive the $C_{11}, C_{12}, C_{21}, C_{22}$ of the result $C = A \times B$.

$$
\begin{aligned}
C_{11} &= m_1 + m_2 - m_4 + m_6 \\
C_{21} &= m_6 + m_7 \\
C_{12} &= m_4 + m_5 \\
C_{22} &= m_2 - m_3 + m_5 - m_7.
\end{aligned}
$$

Show that indeed the C_{ij} are such that

$$
\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}.
\end{aligned}
$$

Problem 6. (60 POINTS)

WORD model with arithmetic operations $+, -, *$, SHIFT operations (45 points).

We will show how to determine whether integer N is an integer power of an integer i.e. there exist two integers $A \geq 2, B \geq 2$ such that $N = A^B$ in "optimal" time. If a pair exists the algorithm outputs A,B otherwise it outputs **OUCH**. Before we describe "optimal" we will give and ask you to fill the details and analyze the performance of several variants of an algorithm(s) whose worst-case running time would be of the form

$$O\left(N^a \cdot (\lg N)^b \cdot (\lg \lg N)^c\right)$$

Constants a, b, c are positive $a \leq 1$ and $b, c \leq 4$.

(a) If N is indeed a perfect power, how large can A, B be? Express your answer(s) as a function of N only. **(10 points)**

(b) Dr I.M DUMB suggested the following solution for the problem. What is its running time ? Consider the case of an efficient implementation of line 5. (Ignore Karatsuba.) **(10 points)**

```
  DUMB(N)
1.   AA=N;
2.   BB=N;
3.   for(a=2 ; a <= AA ; a++)
4.      for(b=2 ; b <= BB ; b++)
5.        if ( RaiseToPower(a,b) == N )
6.          print a,b;
7.          return;
8.        endif
9.      endfor
10.  endfor
11.  printline 'OUCH';
```

(c) A CS667 student who solved part (a) advised Dr DUMB into a better set of values for `AA,BB`. Show that this slightly modified version of `DUMB1` can run in time $O\left(N^a \cdot (\lg N)^b \cdot (\lg \lg N)^c\right)$, where $a \leq 2/3$, $b \leq 3/2, c \leq 3/2$. **(10 points)**

(d) A better CS667 student observed that there was a better solution to this problem. Give an algorithm `Better` whose worst-case running time is $O\left(N^a \cdot (\lg N)^b \cdot (\lg \lg N)^c\right)$, where $a < 1/10000$, $b \leq 2, c \leq 2$. **(15 points)**

BIT model.

(e) Show how in the BIT model, part (d) i.e. algorithm `BETTER` can be done in time $O\left(N^a \cdot (\lg N)^b \cdot (\lg \lg N)^c\right)$, where $a = 0, b \leq 4, c = 0$. **Hint:** *Problems 2,3,4 might help, even if you don't solve them!)* **(15 points)**

Problem 7. (60 POINTS)
Strassen vs Cannon's method.

```
// n can be any integer dimension ; i.e. you have to take care and make it
//   a power of two if necessary
// *A, *B, *C are one dimensional arrays of n*n elements (double)
// A[j*n+i] is the i-th row and j-th column element of a two dimensional array
// For Java use one dimensional arrays
Strassen(double *A, double *B, double *C, int n)  //Does Strassen for arbitrary n
ReadMatrix(double **A,int *n, file input-file); //Allocates space for A and reads A
SetMatrix(double *A,double *B,int n); //Allocates space for A and reads A
PrintMatrix(double *A,int n,  file output-file,)//Prints A into file output-file
Cannon(double *A, double *B, double *C, int n);
```

You need to implement the following interface

```
% ./strassen  input-A input-B output-C
or
% java strassen input-A input-B output-C
% ./cannon    input-A input-B output-C
or
% java cannon   input-A input-B output-C
```

where `input-A, input-B` are files containing input matrices A, B and `output-C` is the file that will contain the output $A \times B$ of Strassen's method or the standard $O(n^3)$ Cannon's method. All three files have the same format. The first line contains the dimension n in the form of an integer. Subsequent lines contain in the form of doubles the input elements in row-major format. That is the first $n = 5$ values `1.0 2.0 3.0 4.0 5.0` are the elements of the first row of the 5×5 array. The next 5 values `6.0 7.0 8.0 9.0 and 10.0` are the elements of the second row and so on. Files `input-A, input-B` are read through `ReadMatrix` and file `output-C` is written by `PrintMatrix`. `SetMatrix` allows one to set copy B into A internally.

```
5
1.0 2.0 3.0 4.0 5.0
6.0 7.0
     8.0 9.0 10.0
1.0
  2.0 3.0 4.0 5.0
1.0
  2.0 3.0 4.0 5.0
1.0 2.0 3.0 4.0 5.0
```

Note. The input array(s) can be of dimension say 17×17. After reading such matrices it's up to you to decide how to store such a matrix; Strassen (textbook description) can only deal with 16×16 or 32×32 matrices but not a 17×17 one. When you print the results into `output-C` make sure it is that of a 17×17 matrix and not that of a matrix of some other dimension. For other assumptions, deviations or instructions, provide a `readme.txt` file with your code.

Benchmarking. Time the two method for your choice of $n = 64$, $n = 256$, and $n = 1024$ non-zero non-trivial matrices. Indicate corresponding running times. For Cannon's method also show a MegaFlop rate ($n \times n$ matrix multiplications requires $n^3 + n^2(n-1)$ operations). Strassen should be able to beat Cannon's method for at least one of those cases on your and AFS machines! Or find that n... So 25 of the 75 points will be for benchmarking.

Problem 8. (40 POINTS)
Polynomial Interpolation. Implement Lagrange's interpolation formula with the following interface. File `inputA` has as its first line an integer N. The following N lines have pairs x_i and $f(x_i)$ double values separated by white spaces(s) (eg tabs, spaces), one such pair per line. Find the polynomial f of degree bound N consistent with it. Print the polynomial into a file as indicated by the last command-line argument (in this example `outputA`) in a pretty-printed format, as shown below (eg zero coefficient terms omitted, one multiplicative coefficients omitted, integers without decimal points).

```
% ./lagrange inputA outputA
% java lagrange inputA outputA


% cat input A

3
0.0 1.0
1.0 2.0
2.0 5.0

% cat outputA
 2
x  +  1

% cat outputA_alternative

 x**2 + 1
```