CS 667 : Homework 3(Due: Oct 24, 2005)

Problem 1. (50 points)

(a)

In 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

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

(b)

Solve the recurrence for the number of operations in Strassen's method exactly, i.e. solve

$$M(n) = 7M(n/2) + (18/4)n^2.$$

What is the base case? Explain. How many operations are required for the base case? Explain. Does it matter whether one chooses the base case to be $n = 1$ or $n = 2$?

What happens for the questions above if one uses the modification of the textbook with the recurrence becoming

$$M(n) = 7M(n/2) + (15/4)n^2.$$

Problem 2. (50 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 complexity: i.e. an $M(n)$ matrix multiplication algorithm implies an $O(M(n))$ squaring algorithm and and an $S(n)$ squaring algorithms implies an $O(S(n))$ matrix multiplication algorithm.

b. The Fibonacci sequence is given by the following recurrence $F_{n+1} = F_n + F_{n-1}$ for $n \geq 1$ and $F_0 = F_1 = 1$.

(i) Show how to compute F_n in $O(n)$ time.

(ii) Given an $n \times n$ matrix A show how you can find A^n in $O(n^3 \lg n)$ time.

(iii) Can you improve the obvious time bound in (a)? In particular prove that F_n can be computed in $O(\lg n)$ time.

Hint: You may need to use the result of part (b), i.e. formulate the F_n as a matrix problem. The discussion on page 902 and 903 (Problem section at the end of the Chapter on Number-Theoretic Algorithms may offer you some insight).

Problem 3. (50 POINTS)

NJIT decides to introduce its own coinage with three different types of coins: 1 cent, 5 cent, and 6 cents. We would like to know what is the minimum number of coins we can use if we pay for an item worth n cents. Give an efficient algorithm that if given n as input, it prints as output the minimum set of coins that has value exactly n. Analyze the time and space requirements of your algorithm. Prove its correctness. For example, you can pay an item worth 30 cents by giving five 6-cent coins; other alternatives is six 5-cent coints, or thirty 1-cent coins, or say three 6-cent, two 5-cent, and two 1-cent coins.

Problem 4. (50 POINTS)

(a) Show that any sorting network must have $\Omega(\lg n)$ depth, and $\Omega(n \lg n)$ comparators.

(b) Then show that any sorting network must have depth at least $\lg n$.

Note: In (a) we don't care about constants; in (b) we do care.

Problem 5. (50 POINTS)

(a) Show that an n-input sorting network must contain at least one comparator between the i-th and the $i+1$-st wire for all i such that $1 \leq i \leq n-1$.

(b) Give a sorting network that sorts 4 keys with 5 comparators.

Problem P1. (120-150 points)

This Problem can only be done in C or C++ because of the interface of the `qsort` function. If you do it you get a maximum of 120 points and you enter a contest with other fellow students who do this problem. If there is no competition and YOUR CODE WORKS CORRECTLY on all input instances then you automatically get the remaining 30 bonus points for a total of 150 points. Benchmark or final-code testing programs won't be divulged. Debugging code is available through the CS 435 web-page at

`http://www.cs.njit.edu/ alexg/courses/cs435/handouts.html` section B4.

a. Implement a randomized quick-sort `ranqsort` function with the interface of the Standard C library qsort function. For the syntax of qsort do `man qsort` on afs. In Visual C++ do a `Help | Search` for `qsort`. You determine the details (eg. if it will look like implementation (b) or (c) below or something else.) The interface of your quick-sort would thus be.

```
void ranqsort(void *base, size_t nel, size_t width,int (*compar)(const void *, const void *)) ;
```

b. Implement the quick-sort `quicksort` function with the interface of the Standard C library qsort function that is described on page 146 of CLRS.

```
void quicksort(void *base, size_t nel, size_t width,int (*compar)(const void *, const void *)) ;
```

c. Implement the quick-sort `hoaresort` function with the interface of the Standard C library qsort function that is described on page 160 of CLRS (Problem 7-1).

```
void hoaresort(void *base, size_t nel, size_t width,int (*compar)(const void *, const void *)) ;
```

d. Implement a sorter based on bitonic sorting. It must be able to handle key sizes other than power of two.

```
void bitonicsort(void *base, size_t nel, size_t width,int (*compar)(const void *, const void *)) ;
```

Problem P2. (120 points)

Strassen.

```
// 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 (floats)
// 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(float *A, float *B, float *C, int n)  //Does Strassen for arbitrary n
ReadMatrix(float **A,int *n, file input-file); //Allocates space for A and reads A
SetMatrix(float *A,float *B,int n); //Allocates space for A and reads A
PrintMatrix(float *A,int n,  file output-file,)//Prints A into file output-file
```

You need to implement the following interface

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