CS 667 : Homework 2(Due: Oct 10, 2005)

Problem 1. (50 points)

Evaluate the following polynomial of degree n, at $x = c$ using $o(n)$ additions/subtractions and $o(n)$ multiplications (you are not allowed to use divisions or any operations other than $+, -, *$).

$$f(x) = \sum_{k=0}^{n} \binom{n}{k} x^k$$

[Note:

$f(n) = o(g(n))$ if and only if $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\binom{n}{k} = n!/(k!(n-k)!)$
$n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$].

Problem 2. (50 points)

Evaluating a polynomial A(x) of degree-bound n at a given point x_0 can also be done by dividing $A(x)$ by the polynomial $(x - x_0)$ to obtain the quotient polynomial $q(x)$ of degree-bound $n - 1$ and a remainder r, such that

$$A(x) = q(x)(x - x_0) + r$$

Clearly $A(x_0) = q(x_0)(x_0 - x_0) + r = r$. Show how to compute the remainder r and the coefficients of $q(x)$ in time $\Theta(n)$ from x_0 and the coefficients $a_0, a_1, \ldots,$ of A.

Problem 3. (50 points)

a. What is the largest k such that if you can multiply 3×3 matrices using k multiplications (not assuming commutativity of multiplication), then you can multiply $n \times n$ matrices in time $o(n^{\lg 7})$? What would the running time of this algorithm be?

b. V. Pan has discovered a way of multiplying 68×68 matrices using 132464 multiplications, 70×70 matrices using 143640 multiplications, 72×72 matrices using 155424 multiplications. Which method yields the best asymtotic running time when used in a divide and conquer matrix multiplication algorithm? How does it compare to Strassen's algorithm?

Problem 4. (50 points)

Show that interpolation can be done in $O(n^2)$ time.

Hint: Read Exercise 30.1-5 of CLRS on page 830 (or Exercise 32.1-4 of CLR on page 783), and think of the implications of Problem 2 of this homework and Problem 4 of HW1.

Problem 5. (50 points)

Go to page 844 of CLRS. Problem 30-1 was partly solved in class; part (a) in the context of complex numbers, and part (c) through the Karatsuba-Ofman algorithm. Do part (b).

Problem 6. (Bonus 50 points)

You are given three polynomials $f(x), g(x), h(x)$ of degree bounds $n+1$, $n+1$, and $2n+1$ respectively. We would like to verify whether $f(x)g(x) = h(x)$ very fast but not with absolute certainty. The verification algorithm does not need to be deterministic; it can be a randomized algorithm.

One naive approach is to find the product $f(x)g(x)$ and compare it to $h(x)$. Finding the product takes $O(n^2)$ time (or $O(n \lg n)$ using FFT-based methods that will be introduced later in this course) and comparison of $f(x)g(x)$ and $h(x)$ takes $O(n)$ time. The total amount of time is, however, $\omega(n)$.

Give a randomized algorithm that determines in $O(n)$ time whether $h(x)$ is $f(x)g(x)$ with confidence at least $1 - 1/n^{10}$. In addition the number of random bits you use should not exceed $O(\lg n)$. For example if you want to draw a random number in the interval $[1, N]$ it will cost you $\lg N$ bits because the largest such number requires about $\lg N$ bits to be represented.

Problem P1. (100 points)

Implement Shamir's secrete sharing scheme.

```
ShamirCreate(secret k, parties p, reconstruct  r, file-out file-name)
  // Returns  a file-name that contains one per-line the individual secrets
  // assigned to each of the p parties. file-name thus has p lines one for each party

secret ShamirReconstruct(parties p, reconstruct  r, file-in  file-name)
  // Uses file-name with at least r lines but no more than p to reconstruct
  // the secret s that is returned.

  // Details are left to you for implementation.
```

The interface will be through the command-line. A

```
% ./ShamirCreate k p r out-file
```

will call the corresponding function and generate some output in file out-file. (Note that `ShamirCreate` is not only a function name but also a program name.)

A `ShamirReconstruct p r my-file` will use my-file (containing lines of out-file) to return in the standard output the secret.

The catch of this Problem: Numbers can grow very big! The secret is a positive 32-bit integer `int` or `unsigned int`.

Problem P2. (100 points)

Implement the Karatshuba-Ofman algorithm. 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, int n1);
MultiplyKaratuba(longint d1, int n1,longint d2, int n2, longint d3, int n3);
WriteKaratsuba(longint d1, int n1, file digit1);
```

A file contains on its first line an (in fact two) integers indicating the number of (decimal) digits that will follow. The remaining lines contain the digits. Line breaks, spaces, tab might exist but are ignored. For example

```
10 10
12345 56
789
```

stores integer the 10-digit integer `1234556789`. Note that the length in digits is given twice in the first line. The first integer is indeed the length; the second is to note the decimal-point in the case that the number is not an integer. Thus viewing the input as real number we get `1234556789.` by including a decimal point to the right of the 10-th digits (the second 10 of the first line).

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

```
% ./karatshuba  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 decimal digits per line.