Problem Set 5: Due by December 4, 2006

Problem 1. (50 points)

The prime number theorem (Theorem 31.37) states that the number of primes less than x is about $x/\ln x$. Therefore the density of primes is $1/\ln x$. In other words if we pick uniformly at random an integer z between 1 and N with probability $1/\ln N$ we choose a prime number and with probability $1-1/\ln N$ we choose a composite number. Let $P = P_m \ldots P_1$ be an m-bit integer. Let $Q = Q_m \ldots T_1$ be an m-bit integer. Then $P = P_m 2^{m-1} + \ldots + P_1$ and $Q = Q_m 2^{m-1} + \ldots + Q_1$.

(a) Is $P \leq 2^m$? Explain. Is $Q \leq 2^m$? Explain.

(b) If $p/(P-Q)$ then show that $P \bmod p \equiv Q \bmod p$. Then show that if $P \bmod p \equiv Q \bmod p$, the $p/(P-Q)$.

(c) Show P or Q can have at most m distinct prime divisors.

(d) Let us pick a random number prime number p between 1 and $R = 4n^2 m \ln 4n^2 m$, where $n > m$. Show that such a p divides $P - Q$ with probability at most $O(1/n^2)$.

In Rabin-Karp let P be the pattern of length m and T be the text of length n. Choose prime R so that $R = \Theta(4n^2 m \ln(4n^2 m))$; R is the prime number used to perform fingerprint computations modulo R.

(e) Show that the probability a spurious hit occurs in position i of the text is $O(1/n^2)$.

(f) Show that the probability a spurious hit occurs anywhere in the text is $O(1/n)$.

(g) Show that the expected running time of the algorithm is $O(n)$.

Problem 2. (40 points)

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 or give the algorithm. You can cite results of the notes/textbook.

Problem 3. (40 points)

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, i.e. complete the proof of Corollary 5 of the notes.

Hint. Use Corollary 4. Be very careful in counting operations. Go through all the division steps and count the bit operations individually and be careful not to overestimate them. Treat the first division step separately.

Problem 4. (40 points)

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, i.e. the time depends on the number of bits of the result, not the number of bits of the terms or the intermediate products.

Note: First solve this problem and then try to do 5,6b.

Problem 5. (40 points)

Given integers a and b how long (in bit, NOT word operations) does it take to efficiently compute integer $n = a^b$, i.e. raise a to the power of b? Express the number of bit operations required in asymptotic notation as a function of n in an efficient solution to this problem. **Note**. An answer of the form: "*At most* $2\lg n - 1$ *multiplications*" is not suitable as an answer to this problem.

Problem 6. (40 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. Points assigned will depend on the amount of bit operations performed.

Programming Problem 1. (60 points)

Implement the algorithm for the perfect power problem (see Solutions of Problem 1 of PS 1) in C or C++.However A can be arbitrarily long (eg. 1024 or 8192 bits). You are allowed to use libraries for arbitrary precision arithmetic as long as (a) they are for free (i.e. I can use them and install them on a linux machine to test your code) and (b) they are easily installable (i.e. I can install them easily). Alternatively, you can implement your own functions for auxiliary operations (eg. arbitrarily long multiplication, exponentiation, etc).

I expect as an answer a .tar file sent by email. The .tar will be untarred on a Red Hat linux workstation and compiled through gcc. I expect a make install command to compile everything and create an executable file named powertest. powertest will accept as input a file containing n in decimal notation.

Thus if file myfile contains a base-10 integer such as

```
17487686712733928413644063750880864826316326531082890839798047131393
06920507121153268532108269241880671097659463948848837902966613039193
61801624726151915125942668649508993665419986623407409256320591797254
65901781768127877188903984695217669171609482765052925389918678373962
03339268758977282743385306120289869213112851446870454351518286400633
04862801177174173384780775389699560332291136389670468588940721006781
36076427022136733286307364152390825026213339153406940132529505642288
77265570344122667987101745048846965145 6
```

then the following command should return

```
% powertest myfile
  x= 123456 y=101
  It's a power!
```

within a reasonable amount of time (eg. under 60 seconds for integers as long as 8192 bits, i.e with up to 2000-3000 digits). `myfile` is a string corresponding to an arbitrarily-named file (in this instance `myfile`).

Programming Problem 2. (60 points)

Implement the NaiveAlgorithm, RabinKarp, BoyerMoore, and KMP by providing C/C++ implementations **consistent with the descriptions of these algorithms provided in class and in the available notes**. For example implementing descriptions or pseudocode for RabinKarp or any of the other algorithms found on the Web will be rejected. String searches should be case insensitive. Then test, your algorithms on texts available at the Project Gutenberg web-site. `http://promo.net/pg/`. Some texts that you should test your algorithms on are

```
1. US Copyright Law at
   ftp://ftp.archive.org/pub/etext/etext03/clusa10.txt

2. George Bernard Shaw's  "Caesar and Cleopatra".
   ftp://ftp.archive.org/pub/etext/etext02/candc10.txt  .

3. Orlando Furioso by Ludovico Ariosto at
   ftp://ftp.archive.org/pub/etext/etext03/8ofur10.txt  .

4. Human Genome Project, Chromosome 1,
   ftp://ftp.archive.org/pub/etext/etext00/01hgp10.txt
```

Search 4-letter, 8-letter and 16-letter strings that appear and do not appear in the text and report the running times for finding all occurrences in a tabular form and the number of successful hits. In addition search the strings "notwithstanding" and "protection" in the first text, "Caesar" and "Cleopatra" in the second, and "Alessandro" and "Alcina" in the third, and "AGCTAGCT" and "AAAAGGGCC" in the fourth, and report the number of hits in addition to running times.

You need to provide the following interface

```
% java  strmatch   XXX file-name


 or
%  ./strmatch XXX file-name  pattern
```

`XXX` is one of `NAL, KMP, RK, BM` and indicates the corresponding algorithm i.e. the naive, Knuth-Morris-Pratt, Rabik-Karp or Boyer-Moore. Parameter `file-name` is an arbitrary string that indicates the file that will be used as input. Parameter `pattern` is the string that will be searched in `file-name`. The output should be of the following form.

```
 No instance of  pattern was found in file-name
```

or

```
 1. An instance of pattern was found at position 19
 2. An instance of pattern was found at position 192
 3. An instance of pattern was found at position 1922

 A total of 3 instances of pattern were found.
```

depending on whether `pattern` exists or not in `file-name`.