Computer Science 308-250B Homework #2
Due Monday February 9, 2004, 13:30

In all the problems, **<u>all calculations are done mod M</u>** ***where*** **M=3333373** ***in order to limit the size of the integers involved. This way we avoid problems due to overflow.***

As we have seen in class, the Fibonacci numbers are defined as follows:

$$f_n = \begin{cases} 1 & \text{if } n=1,2 \\ f_{n-1} + f_{n-2} & \text{if } n>2 \end{cases}$$

Thus the most natural way to write a program to compute one uses the next algorithm:

```
fibrec(n)
If (n<3) Then return 1
Else return fibrec (n-1)+ fibrec (n-2)
```

[15%] •**1)** Write a **Java** program that implements this algorithm. Find the time needed to compute $f_1, f_2, f_3, f_4, f_5, \ldots$ and plot a graph of your results. What is the largest n for which you can compute f_n within 1 second ?

An iterative way of computing the same values only keeps track of the last two values and use them to compute the new ones :

```
fibit(n)
f:=1; g:=1
For k:=2 to n do
        f:= f+g
        g:= f-g
Return f
```

[15%] **•2)** Write a **Java** program that implements this algorithm. Find the time needed to compute $f_1, f_2, f_3, f_4, f_5, \ldots$ and plot a graph of your results. What is the largest n for which you can compute f_n within 1 second ?

An alternate definition we have **NOT** seen in class for the Fibonacci numbers is:

$$f_n = \begin{cases} 1 & \textbf{if n=1,2} \\ f^2_{(n+1)/2} + f^2_{(n-1)/2} & \textbf{if n>2, n odd} \\ f^2_{n/2+1} - f^2_{n/2-1} & \textbf{if n>2, n even} \end{cases}$$

[15%] **•3a)** Prove by mathematical induction that this new definition of f_n is correct.

[15%] **•3b)** Write a **Java** program that implements this algorithm. Find the time needed to compute $f_1, f_2, f_3, f_4, f_5, \ldots$ and plot a graph of your results. What is the largest n for which you can compute f_n within 1 second ?

[15%] **•4a)** Prove the following by mathematical induction for all **n>0**.

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n} = \begin{pmatrix} f_{n-1} & f_{n} \\ f_{n} & f_{n+1} \end{pmatrix}$$

A natural way of computing this exponentiation is to use a method similar to the one we used in Homework #1 for numbers (below **I** stands for the 2 by 2 identity matrix):

```
ExpMOD(A,b)
If (b=0)
Then return I
Else  If (b mod 2)=0   Then return ExpMOD( A^2, b/2 )
                       Else return A*( ExpMOD(A^2, (b-1)/2 ) )
```

NOTE: when you implement matrix multiplication make sure all your coeeficients are computed mod M.

[15%] **•4b)** Write a Java program that implements this algorithm. Find the time needed to compute $f_1,f_2,f_3,f_4,f_5,\ldots$ and plot a graph of your results. What is the largest n for which you can compute f_n within 1 second (give an estimate if n is too big…)?

[10%] **•5)** If **M** is rather small (say M<1000) and you wish to compute the value $\mathbf{f_n}$ **mod M**; find a very fast algorithm to compute this value, even for very large values of **n**.
hint: Show that there exists an integer R, $0<R<M^2$ such that $\mathbf{f_n}$ **mod M** = $\mathbf{f_{n \bmod R}}$ **mod M.**

<u>About counting time…</u>

In **Java** you may find out what the time is by calling the method

System.currentTimeMillis()

Here is what the **Java** documentation has to say about this method:

20.18 The Class java.lang.System
…
public final class System {
…
public static long currentTimeMillis();
…
}

20.18.6 public static long currentTimeMillis()

Returns the difference, measured in milliseconds, between the current time and the standard base time known as "the epoch," 00:00:00 GMT on January 1, 1970. See the description of the class Date (§21.3) for a discussion of slight discrepancies that may arise between "computer time" and UTC (Coordinated Universal Time).

• **<u>You must realize that calling this method takes time</u>,** thus you have to be careful in the way you use it. For instance, don't use it in the middle of another method you want to time. You should call **System.currentTimeMillis()** just before and right after the call to the other method you are timing.

• If the time required to complete a calculation is **too small** you may have to run it several times in a loop and calculate the total running time divided by the number of trials.

• If the time required to complete a calculation grows **too slowly** you may choose to calculate values at a larger interval. Computing $f_1,f_2,f_4,f_8,f_{16},\ldots$ may be more significant in some cases.