GENERATING FIBONACCI NUMBERS

Problem Statement: Find $f(10^9) mod (10^9 + 7)$

Before proceeding further I would request people who haven't seen these symbols before in their lives to ignore them totally and read this article as the math I'm going to use in this, is very basic but the ideas are powerful. The objective of the problem is to find the billionth fibonacci number, and as it has more than 200 million digits in it, we are expected to output the remainder which remains after this huge number is divided by $10^9 + 7$. Nothing special about $10^9 + 7$ other than the thing that it is a prime. And it is usually used in many contests as $C++$ has int data type which holds numbers between -2^{31} to $2^{31} - 1$ and this number fortunately lies in this range and is also good looking.

Introduction: Fibonacci Numbers are generated by this rule.
$f(n) = f(n-1) + f(n-2)$ for $n \geq 2$
where, $f(0) = 0$ and $f(1) = 1$

So how would you go about finding $f(10^9)$??
Lets us explore all possible ways of doing it.
Note: Here on I'll use f_n instead of $f(n)$ as the former is a lot easier to type in latex.
1. The iterative way: As the name suggests we go one step at a time. First we start with $n = 2$, we compute it using f_0 and f_1 then we go about finding f_3 then f_4 and so on. Let us crudely examine how much time it would take for the program to find f_n. A simple analysis says that we need n steps to find f_n as is clear from the algorithm. Now assume that a computer can do a million such steps in one second then our program would approximately take a 1000 seconds to puke a result. Is it not fascinating that we could find the billionth fibonacci number within 15 mins?? Maybe it is fascinating for a person who doesn't know how powerful computers are, but for a person who has little bit of technical knowledge will not appreciate this algorithm because if instead we want to find $f_{10^{10}}$ we would have to wait for almost 2.5 hrs for the result. So the big question is how to find such large numbers quickly?
Before going further do have a look at the program(fibiter.cpp) which implements this algorithm and test the speed for various inputs yourself.
2. The elegant way: The code for the previous algorithm looked quite lengthy for a supposedly simply iteration. So Mr. Stupid decides to code elegantly. So he uses a recursive function to evaluate f_n which calls f_{n-1} and f_{n-2} recursively till they hit 1. I have also attached the code(fibrecursive.cpp) for the above algorithm. The reason for Mr. Stupid's name to be stupid will be quite clear when you try to find f_n for $n \geq 30$.
Note: The person who first points out the reason why stupid's algorithm is slow(damn slow) will recieve special appreciation. And the person who suggests a way to speed up the algorithm using recursion itself will get a chocolate.
3. The super-fast way: Now everyone must be excited to know this super-fast way but I won't give you the whole code rightaway as this technique needs to be

appreciated properly and so I'll discuss this technique over this week, a bit each day. Actually there are two different techniques. One is based on this expression for fibonacci numbers $f_{n+m} = f_m f_{n+1} + f_{m-1} f_n \ \forall m \geq 1$
If I give away the next hint then all fun is gone but still I would give a partial hint. Put $n = m$ and $n = m - 1$ in the above expression and you'll see the magic. Another method is based on fast exponentiation. I will surely introduce it properly within a day or two so that any person with just a basic understanding of matrices and binary representations will be able to implement it. For those who know how to implement fast exponentiation here is the hint.

$$\begin{bmatrix} f_{n+1} \\ f_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_n \\ f_{n-1} \end{bmatrix}$$

So we have,

$$\begin{bmatrix} f_{n+1} \\ f_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} f_1(=1) \\ f_0(=0) \end{bmatrix}$$

Now, I think some of you will find your way on your own.