Method 6 (O(Log n) Time)

Below is one more interesting recurrence formula that can be used to find n'th Fibonacci Number in O(Log n) time.

```
If n is even then k = n/2:
F(n) = [2*F(k-1) + F(k)]*F(k)

If n is odd then k = (n + 1)/2
F(n) = F(k)*F(k) + F(k-1)*F(k-1)
```

How does this formula work?

The formula can be derived from above matrix equation.

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

Taking determinant on both sides, we get

$(-1)^n = F_{n+1}F_{n-1} - F_n^2$

Moreover, since $A^nA^m = A^{n+m}$ for any square matrix A, the following identities can be derived (they are obtained form two different coefficients of the matrix product)

$F_mF_n + F_{m-1}F_{n-1} = F_{m+n-1}$

By putting n = n+1,

$F_mF_{n+1} + F_{m-1}F_n = F_{m+n}$

Putting m = n

$F_{2n-1} = F_n^2 + F_{n-1}^2$

$F_{2n} = (F_{n-1} + F_{n+1})F_n = (2F_{n-1} + F_n)F_n$ (Source: Wiki)

To get the formula to be proved, we simply need to do following

If n is even, we can put k = n/2

If n is odd, we can put k = (n+1)/2

Below is C++ implementation of above idea.

```
// C++ Program to find n'th fibonacci Number in
// with O(Log n) arithmatic operations
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1000;

// Create an array for memoization
int f[MAX] = {0};

// Returns n'th fuibonacci number using table f[]
int fib(int n)
{
    // Base cases
    if (n == 0)
        return 0;
    if (n == 1 || n == 2)
        return (f[n] = 1);

    // If fib(n) is already computed
    if (f[n])
        return f[n];

    int k = (n & 1)? (n+1)/2 : n/2;

    // Applyting above formula [Note value n&1 is 1
    // if n is odd, else 0.
    f[n] = (n & 1)? (fib(k)*fib(k) + fib(k-1)*fib(k-1))
           : (2*fib(k-1) + fib(k))*fib(k);

    return f[n];
}

/* Driver program to test above function */
int main()
{
    int n = 9;
    printf("%d ", fib(n));
    return 0;
}
```