递归做法 O(2^n)

```
//斐波那契递归做法
#include <bits/stdc++.h>
using namespace std;
int f(int x)
{
    if(x==0 || x==1)
        return x;
    else return f(x-1)+f(x-2);
}
int main()
{
    int n;
    cin >> n;
    cout << f(n);
}
```

递推做法 O(n)

```
//斐波那契递推做法
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6+20;
int f[N];
int main()
{
    f[0]=0;f[1]=1;
    int n;
    cin >> n;
    for(int i=2;i<=n;i++)
        f[i]=f[i-1]+f[i-2];
    cout << f[n];
}
```

矩阵快速幂 O(log_n)

```
//快速幂 -- 为矩阵快速幂算法做铺垫
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int a,b;//求a的b次幂
    cin >> a >> b;
    //第一种方法:原始迭代
    int res = 1;
    int bb = b;//防止b自减,保护b数据
    while(bb--)//循环b次
        res*=a;
    cout << res << endl;
```

```
    //第二种方法:调用库函数
    cout << pow(a,b) << endl;//库函数在math.h
    //第三种方法:快速幂
    res = 1;
    //假设a是3,b是11,b在计算机中存储为1011
    //a^b便是3^11
    //可以用二进制化为3^(1* 2^0 + 1* 2^1 + 0* 2^2 + 1* 2^3)
    //继续化简为 3^(1* 2^0 ) * 3^(1 * 2^1) * 3^(0 * 2^2) * 3^(1 * 2^3)
    while(b)
    {
        if(b&1)//让末位数和1做和运算,目的是判断末尾数字是否为1,很优雅的代码
                //第一轮提取出最后一个位，开始是1
                //第二轮提取出新的最后一个位，是1
            res*=a;//3^(1* 2^0) //3^(1* 2^1)
        //去掉尾数
        b >>= 1;//把末位数去掉
        //每次要相乘的数都是前一个数的2倍
        a*=a;//开始是3^1,然后是3^2,每次都是前一次的平方倍
    }
    cout << res << endl;
}
```

$0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad n$
$0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad f(n)$

原本一项与前两项有关，现将上述规律扩展至二元，每一项只和前一项有关。

$$\begin{pmatrix}0\\1\end{pmatrix} \quad \begin{pmatrix}1\\1\end{pmatrix} \quad \begin{pmatrix}1\\2\end{pmatrix} \quad \begin{pmatrix}2\\3\end{pmatrix} \quad \begin{pmatrix}3\\5\end{pmatrix} \quad \begin{pmatrix}5\\8\end{pmatrix} \quad \begin{pmatrix}f(n)\\f(n+1)\end{pmatrix}$$

$$A^{\circ} \cdot B \quad A^1 \cdot B \quad A^2 \cdot B \quad A^3 \cdot B \quad \cdots \quad A^n \cdot B$$

$$A \cdot \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}1\\1\end{pmatrix}$$

$$A = \begin{pmatrix}0 & 1\\1 & 1\end{pmatrix}$$

得到通项公式$A^n \cdot B$，要求的便是此矩阵的第一个元素。

由此可以将一个斐波那契问题转换成快速幂问题，及如何快速求得A^n

```
//斐波那契 矩阵快速幂做法
//无需掌握，只需要了解下
//我上课说的推导流程比代码更重要
#include <bits/stdc++.h>
using namespace std;
const int mod = 1e9 + 7;//数字过大进行取模

//matrix n矩阵
struct mat
{
    //五行五列二维数组(数组开的大一些)
    long long a[5][5];
```

```
};

//两个矩阵相乘,返回矩阵
mat mul(mat a,mat b)
{
    mat ans;//结果
    memset(ans.a,0,sizeof ans.a);//初始化:ans为mat类型的结构体,ans.a为矩阵
    for (int i = 0; i < 2;i++)
        for (int j = 0;j < 2;j++)
            for (int k = 0;k < 2;k++)
                ans.a[i][j] += (a.a[i][k] * b.a[k][j]) % mod;//a和b为mat类型的结构
体,a.a和b.a为矩阵
    return ans;
}

//n代表指数
long long quick_power(long long n)
{
    mat ans,res;//ans为结果矩阵,res为系数矩阵

    memset(ans.a,0,sizeof ans.a);//初始化结果矩阵
    ans.a[0][0] = 1;
    ans.a[1][0] = 1;

    memset(res.a,0,sizeof res.a);//初始化系数矩阵
    res.a[0][0] = 1;
    res.a[0][1] = 1;
    res.a[1][0] = 1;
    res.a[1][1] = 0;

    //核心代码,快速幂算法
    while (n)
    {
        if (n & 1) ans = mul(res,ans);//res和ans两个矩阵不能颠倒
        n >>= 1;//指数减半
        res = mul(res,res);//底数平方
    }
    return ans.a[0][0];//ans.a[0][0]即图中的f(n)
}

int main()
{
    int n;
    cin >> n;
    //1 1 2 3 5 8 13 21 34
    if (n == 1 || n == 2) cout << 1 << endl;
    else cout << quick_power(n - 2) << endl;
    return 0;
}
```