1. 大整数乘法递归代码：

```
#include <iostream>
using namespace std;

int f(int x, int y, int n)
{
 int result, a, b, c, d, ac, bd, S;

 if (n == 1)
       return x * y;
 a = x >> (n / 2);
 b = x - (a << (n / 2));
 c = y >> (n / 2);
 d = y - (c << (n / 2));
 ac = f(a, c, n / 2);
 bd = f(b, d, n / 2);
 S = f(a - b, d - c, n / 2);
 result = (ac << n) + ((S + ac + bd) << n / 2) + bd;

 return result;
}

int main()
{
 int x, y;

 cin >> x;
 cin >> y;
 cout << f(x, y, 32);

 return 0;
}
```

2. 汉诺塔有没有效率更高的算法：

经过分析，汉诺塔的移动最小步数和层数的关系为 2 的幂次关系，原有算法已经是这个效率了，因此不可能从减少移动步数的角度进行优化，我尝试采用非递归算法的方法进行优化，因为递归涉及数据的保存和回复等操作，我猜测使用非递归的方法会省去这些开销。

递归算法代码：

```
void hanoi(char a,char b,char c,int n)
{
 if(n>1)
       hanoi(a, c, b, n - 1);
 cout<< n << ':' << a << "->" << c << endl;
 if(n>1)
```

```
        hanoi(b, a, c, n - 1);
    }

    int main()
    {
     char a, b, c;
     int n;

     a = 'a';
     b = 'b';
     c = 'c';
     n = 30;
     hanoi(a, b, c, n);

     return 0;
    }
```

非递归算法代码:

```
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
    char a, b, c;
    int n, steps, coe, tmp;

    a = 'a';
    b = 'b';
    c = 'c';
    n = 30;
    char location[30];
    for(int i=0;i<n;++i)
        location[i]='a';
    steps = pow(2, n) - 1;
    for (int i = 0; i < steps; ++i)
    {
        coe = n - 1;
        while (true)
        {
            int mid = 1;
            mid = mid << coe;
            if ((i + 1) % mid == 0)
                break;
            --coe;
```

```
            }
            cout << coe + 1 << ':' << location[coe] << "->";
            if (n % 2 == 0 && (coe + 1) % 2 != 0)
                tmp = (location[coe] - a + 1) % 3;
            else if (n % 2 == 0 && (coe + 1) % 2 == 0)
                tmp = (location[coe] - a + 2) % 3;
            else if (n % 2 != 0 && (coe + 1) % 2 != 0)
                tmp = (location[coe] - a + 2) % 3;
            else if (n % 2 != 0 && (coe + 1) % 2 == 0)
                tmp = (location[coe] - a + 1) % 3;

            location[coe] = a + tmp;
            cout << location[coe] << endl;
        }

        return 0;
        }
```

经过不断的调整汉诺塔的层数的参数，得到程序运行时间如下（去除输出语句，因为该语句的时间相同且占比较大，不利于观察）

	20	21	22	23	24	25	26	27	28	29	30
递归	0.3632s	0.3745s	0.3633s	0.3593s	0.3529s	0.4528s	0.6073s	0.7595s	1.217s	2.135s	3.84s
非递归	0.3577s	0.4533s	0.5310s	0.7839s	1.302s	2.352s	4.627s	8.833s	15.87s	30.95s	61.43s

将递归和非递归的运行时间加以对比，发现在层数小于 20 层时，随层数增加时间增长不明显，几乎无增长，我认为是因为除开实际和汉诺塔层数有关的部分的时间太短，在整个程序运行时间中占比很小，所以变化不明显。在 20 层以后，随着层数的逐渐增多，和层数有关的程序运行时间增大，变化逐渐明显。

跟我的猜测相反，非递归算法相比于递归算法的时间反而更长，并且随着层数的增加差距更加明显。我猜想对应每一次移动，我所写的非递归算法的冗余步骤比递归算法保存数据等花费的时间更多，虽然省下了递归的一些开销，但增加了其他的运算和开销，所以时间反而更长。

经过思考，我没有想出能够节省下非递归算法的计算步骤的有效方法，因此我目前认为汉诺塔的递归算法很高效。