实验报告

比较计算Fibonacci数列的多种方法

实验环境

CPU：Intel Core i5-7200U

内存：8GB

操作系统：Windows 10 64位教育版

编程语言：Python 3.6

算法分析

Naive Recursive

根据Fibonacci数列的递推公式$F_n = F_{n-1} + F_{n-2}$，递归求值，其时间复杂度$T(n) = \Omega(\phi^n), \phi = \frac{1+\sqrt{5}}{2}$

Naive Recursive Squaring

在一定范围内F_n与$\frac{\phi^n}{\sqrt{5}}(\phi = \frac{1+\sqrt{5}}{2})$非常接近，故可通过计算$\frac{\phi^n}{\sqrt{5}}$来近似估算$F_n$的值，其时间复杂度$T(n) = \Theta(lgn)$

Bottom-up

自下而上地计算F_0到F_n的值，其时间复杂度$T(n) = \Theta(n)$

Recursive squaring

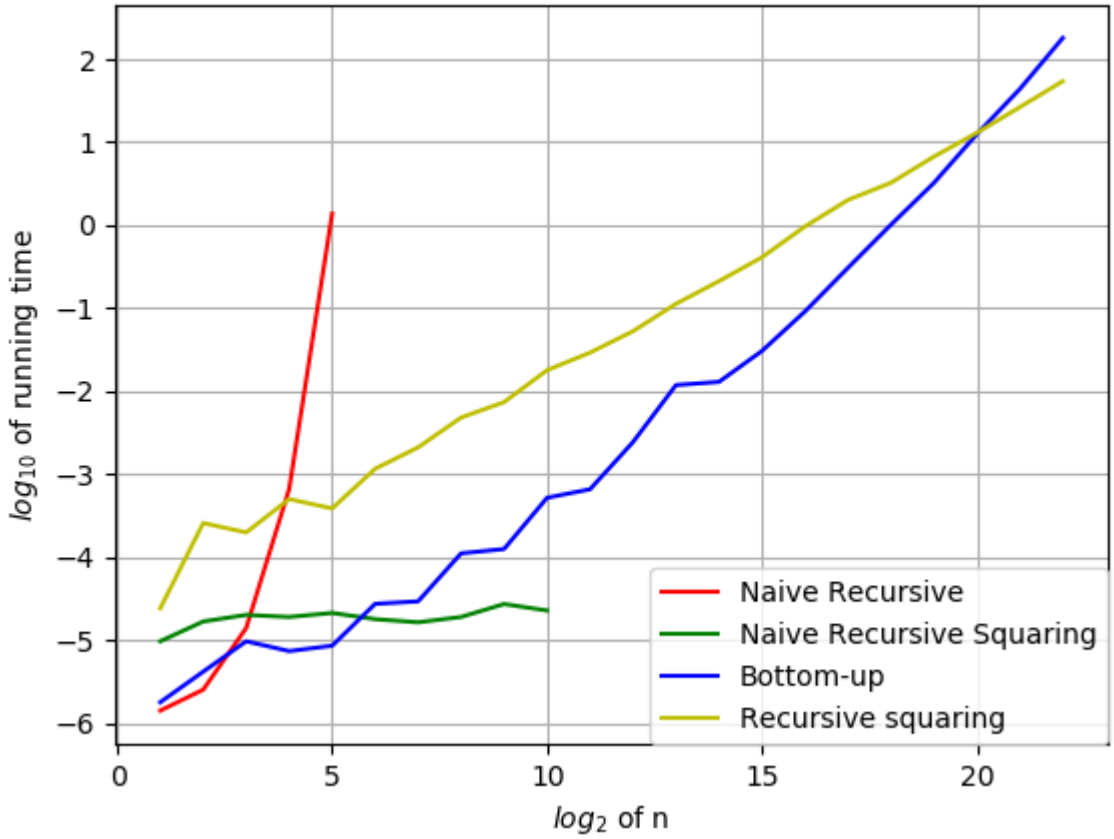
利用矩阵运算计算F_n，时间复杂度$T(n) = \Theta(lgn)$

结果分析

取$n = 2^i$，分别使用4种算法进行实验，每组实验均进行5次取运行时间的平均值，并记录其运行时间和计算结果，运行时间如下表（单位为秒）：

$n=2^i$	Naive Recursive	Naive Recursive Squaring	Bottom-up	Recursive squaring
1	1.43480e-06	9.81709e-06	1.88790e-06	5.55043e-05
2	2.56754e-06	1.69911e-05	4.30442e-06	0.00012
3	1.41970e-05	2.03893e-05	5.36164e-06	0.00028
4	0.00067	1.92566e-05	5.89025e-06	0.00052
5	1.37983	2.15221e-05	1.55563e-05	0.00094
6	严重超时	1.81238e-05	2.08424e-05	0.00168
7		1.66135e-05	3.99480e-05	0.00650
8		1.92566e-05	0.00010	0.00771
9		2.75633e-05	0.00017	0.01446
10		2.30324e-05	0.00057	0.04520
11		浮点数溢出	0.00079	0.05316
12			0.00333	0.09110

由于递归层数和内存限制及浮点数溢出等问题，两种Naive算法仅在n较小时能正常运行，并且随着n的增大Naive Recursive Squaring算法的近似误差也越来越大，在$n=2^{10}$时计算误差已经达到了10^{199}，可靠性不足。因此接下来的实验将单独对Bottom-up和Recursive squaring算法进行比较，运行时间如下图：



当$n < 2^{20}$时，Recursive squaring算法由于内存分配耗费较多时间和$T(n)$中$o(lgn)$项影响的原因，耗时比Bottom-up算法偏多，当$n > 2^{20}$时，$T(n)$的差异使得Recursive squaring算法开始快于Bottom-up算法。

因此可把2^{20}作为本环境下的crosspoint，当$n < 2^{20}$时选择Bottom-up算法，当$n > 2^{20}$时选择Recursive squaring算法，同时当$n < 2^{10}$时也可考虑使用Naive Recursive Squaring算法来计算F_n的近似值。