Programming Project Report 1 – Performance Measurement

Chapter 1: Introduction

There are at least two different algorithms that can compute X^N for a given positive integer N.

Algorithm 1 is to use normal $N - 1$ multiplications, repeatedly multiplying X onto the result.

Algorithm 2 works in the following way:

if N is even, $X^N = X^{N/2} \times X^{N/2}$; and

if N is odd, $X^N = X^{(N-1)/2} \times X^{(N-1)/2} \times X$.

By recursively calling the computing function, we can realize this bisection algorithm.

It is easily seen that these different versions of algorithms would take different levels of time to compute, so in this report we would examine which is more efficient and to what extent one algorithm is better than the other.

Chapter 2: Algorithm Specification

To see if recursion do cause waste of system time, for Algorithm 2 we designed two versions, one recursive, the other iterative.

Algorithm 1 is easy; just by doing multiplying commands to a temporary variable we would have the result. Time Cost O(N).

```
Temporary Variable r = 1;
From 1 to n
   Multiply r by x;
Return r as result.
```

Algorithm 2 in recursive version passes the exponent N and multiplier X, forming a bisecting algorithm structure. Time Cost O(log N).

```
If N is 0
   Return 1 to upper level;
If N is even
   Go to next recursion with exponent N/2 and multiplier X^2, returning
that value
If N is odd
   Go to next recursion with exponent N/2 ((N-1)/2 in fact) and multiplier
X^2, returning that value multiplied by X.
```

Algorithm 2 in iterative version bisects the exponent N itself, forming a 32-bit table that tells the program which algorithm to use when coming to this exponent, and then using a loop to follow the table to rebuild this multiplying process from bottom to top. Time Cost O(log N).

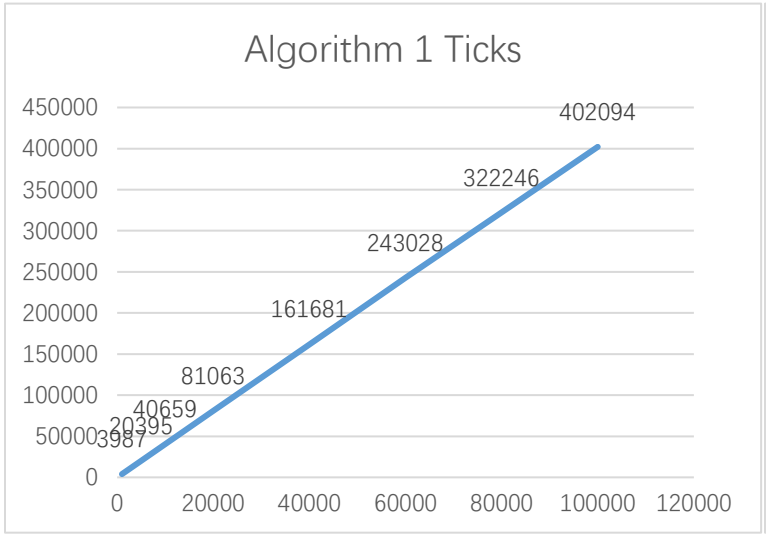
```
Divide N repeatedly by 2, till N becomes zero
   Record its remnant at each point (thus creating a binary version of
N);
Set temporary result r=1;
From 1 to ceiling(log(N))
   If remnant is odd
      r = r(old)^2*x;
   If remnant is even
      r = r(old)^2.
```

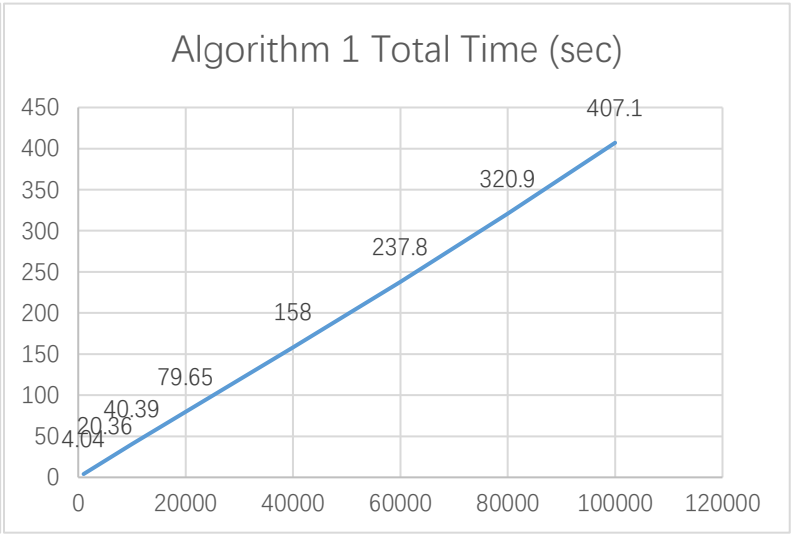
In comparison between versions of Algorithm 2, it seems clear that Algorithm 2 in iterative form avoids redundancy by storing status values in a table and use directly in calculation. The recursive version may waste time, but is a more direct approach.

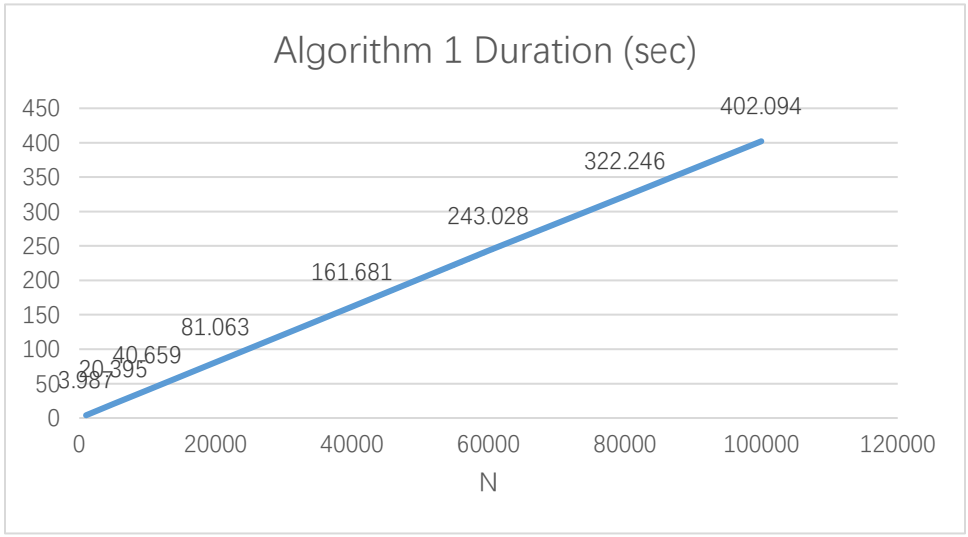
Chapter 3: Testing Results

	N	1000	5000	10000	20000	40000	60000	80000	100000
Algorithm 1	Iterations (*K*)	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000
	Ticks	3987	20395	40659	81063	161681	243028	322246	402094
	Total Time (sec)	3.987	20.395	40.659	81.063	161.681	243.028	322.246	402.094
	Duration (sec)	3.987*10^-6	20.395*10^-6	40.659*10^-6	81.063*10^-6	161.681*10^-6	243.028*10^-6	322.246*10^-6	402.094*10^-6
Algorithm 2 (iterative version)	Iterations (*K*)	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000
	Ticks	522	670	747	805	842	883	909	945
	Total Time (sec)	0.522	0.67	0.747	0.805	0.842	0.883	0.909	0.945
	Duration (sec)	5.22*10^-8	6.7*10^-8	7.47*10^-8	8.05*10^-8	8.42*10^-8	8.83*10^-8	9.09*10^-8	9.45*10^-8
Algorithm 2 (recursive version)	Iterations (*K*)	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000
	Ticks	856	1088	1167	1258	1318	1319	1392	1413
	Total Time (sec)	0.856	1.088	1.167	1.258	1.318	1.31	1.392	1.413
	Duration (sec)	8.561*10^-8	10.889*10^-8	11.67*10^-8	12.584*10^-8	13.186*10^-8	13.19*10^-8	13.926*10^-8	14.138*10^-8

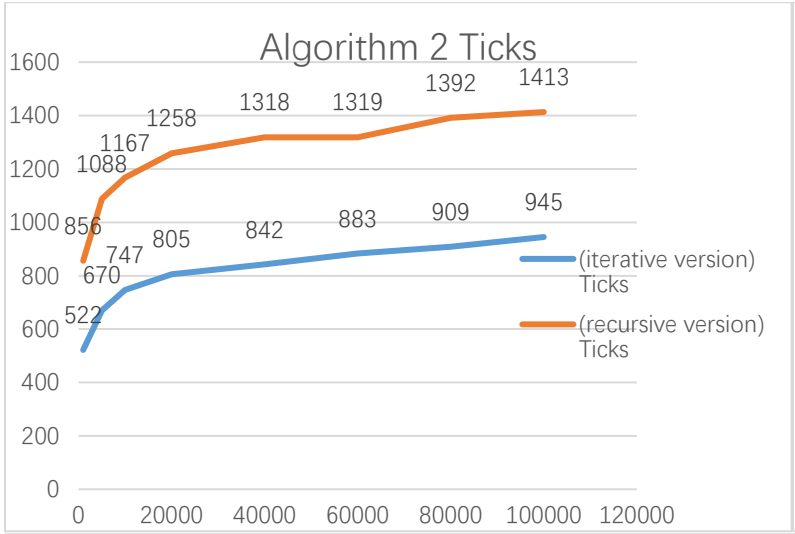
In algorithm 1, K=1000 000.

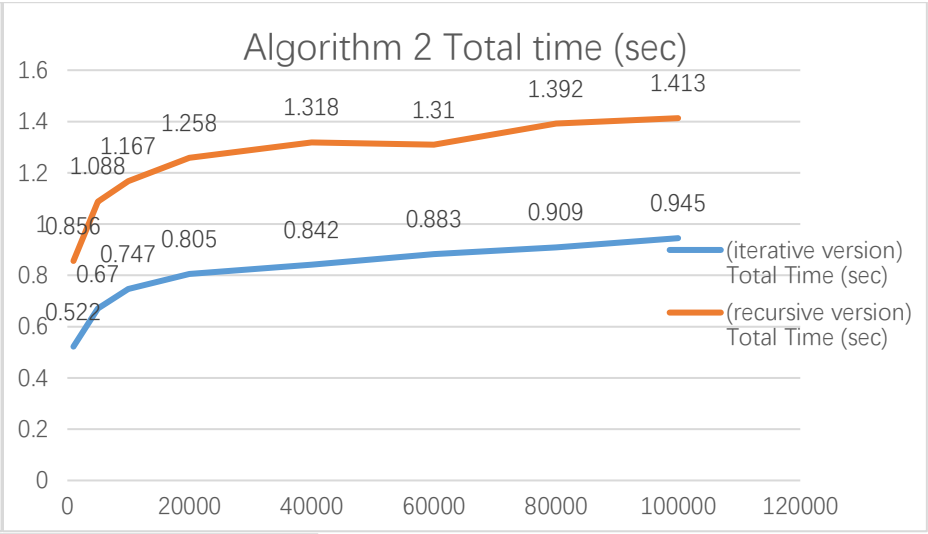


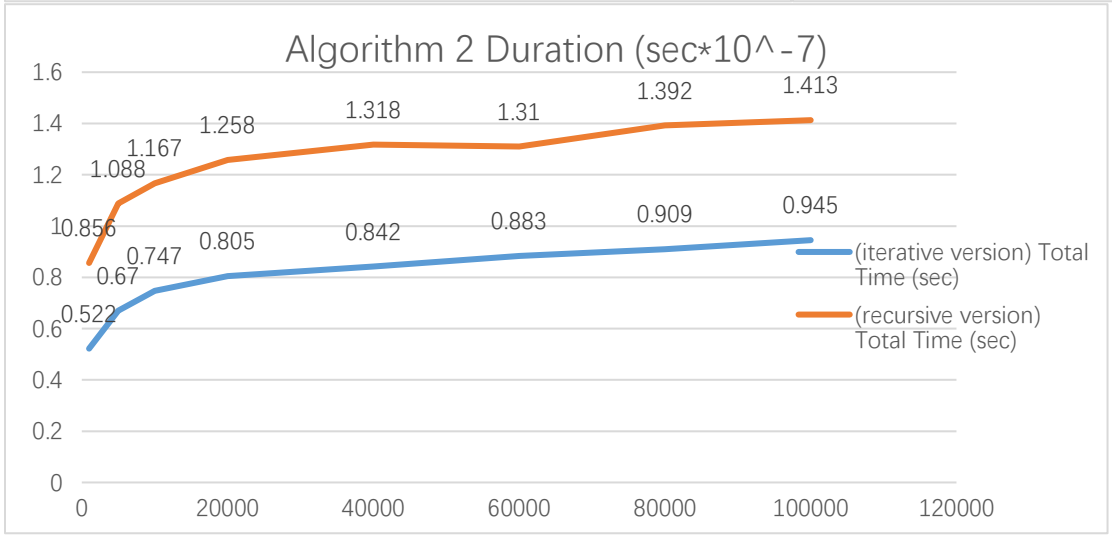




In algorithm 2, K = 10 000 000.







Chapter 4: Analysis and Comments

As we can see about the run times of the functions from the graph, Algorithm 2 is much more efficient than Algorithm 1. From the graph, we can see the Algorithm 1's running time is linear. Both Algorithm 2 (iterative version) and Algorithm 2 (recursive version) have logarithmic time cost growth, but Algorithm 2 (iterative version) is quite more efficient than Algorithm 2 (recursive version).

It has been verified by experiments above that, Algorithm 2 (iterative version)'s time complexity is O(log N), and its space complexity is O(log N). Algorithm 2 (recursive version)'s time complexity is also O(log N), but its space complexity is O(1) in each recursive function, while in total O(log N). Although iterative version's space complexity is larger than recursive version's, iterative version runs quite quicker than recursive version. We can see that the extra time must have been employed to register and clear recursive function records, thus adding to the total running time.

The recursive version's advantages: the code is simple, clear, and easy to verify its correctness. However, it needs more functions to run a function call and takes up more space. But since it only repeats $\log_2(N)$ times, so its shortcoming is not so obvious compared to the traditional O(N) approach. The iterative version's advantage is its simple function calling structure. Even with optimization from the IDE, calling a function still takes time. So it is easily seen that in most times recursive function calls are not recommended in high-efficiency programming.

Appendix: Source Code (in C)

```
/*
2016-17学年 秋冬学期 数据结构基础 冯雁老师
项目1  正数的幂
（小组互评文件中删除）第13组 毛顺龙 陈潼 张岱 出品
We declare that all the work done in this project is of our
independent effort as a group.
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

double pow1_1(double X,unsigned N);   /* 函数声明，依次是algorithm
1、algorithm 2的递归版和循环版 */
double pow2_1(double X,unsigned N);
double pow2_2(double X,unsigned N);

int main()                    /* 主函数、测试函数 */
{
   double X=1.001;
   int N=1000;           /* 测试值X、N与循环次数K，在这里预先初始化了 */
   long long K=10000000;    /* 考虑到32位与64位机器的差别，用了long
long */
   clock_t start,stop;       /* 计算ticks */
```

```
    double duration;          /* 计算用时 */
    double r1=0,r2=0,r3=0;/* 检验结果 */
    int i;                   /* 循环时的辅助变量*/

    printf("请输入X、N和K \n");scanf("%lf%d%ld",&X,&N,&K);

    r1=pow1_1(X,N);
    r2=pow2_1(X,N);
    r3=pow2_2(X,N);
    printf("结果依次是%lf %lf %lf \n",r1,r2,r3);  /* 检验结果是否正确
*/

    /* pow1_1的用时： */
    start = clock();     /* 计时开始 */
    for(i=1;i<=K;i++){    /* 循环重复K次 */
       pow1_1(X,N);
    }
    stop = clock();
    duration = ((double)(stop - start))/CLK_TCK;
    printf("pow1_1的ticks=%d, duration=%lf \n",stop -
start,duration) ;

    /* pow2_1的用时： */
    start = clock();
    for(i=1;i<=K;i++){
    pow2_1(X,N);
    }
    stop = clock();
    duration = ((double)(stop - start))/CLK_TCK;
    printf("pow2_1的ticks=%d, duration=%lf \n",stop -
start,duration) ;

    /* pow2_2的用时： */
    start = clock();
    for(i=1;i<=K;i++){
    pow2_2(X,N);
    }
    stop = clock();
    duration = ((double)(stop - start))/CLK_TCK;
    printf("pow2_2的ticks=%d, duration=%lf \n",stop -
start,duration) ;
```

```
    system("pause") ;
return 1;
}

/*第一个是循环版的累次乘*/
double pow1_1(double X,unsigned N)
{
    int n;              /*为防止在函数中破坏N而引入的辅助变量*/
    double result=1;  /*储存结果，初始化为1 。若N=0 则直接会返回*/
    for(n=N;n>=1;n--){
        result= X*result;
    }
    return result;
}

/*第二个是递归版的“折半法”求幂，算法见教材的2.4.4节*/
double pow2_1(double X,unsigned N)
{
    if(N==0)return 1;   /*递归出口 */
    if(N%2==0)return pow2_1(X*X,N/2);   /*如果是偶数，计算
X^(N/2)*X^(N/2) */
    else return (X*pow2_1(X*X,N/2));    /*如果是奇数，计算
X^(N/2)*X^(N/2)*X */
}

/*第三个是循环版的“折半法”求幂，算法思想详见项目文档 */
 double pow2_2(double X,unsigned N)
{
    double result=1;   /*最终结果，初始化为1 */
    short int l[32]={0};
     /*根据算法思想，需逆向求解，因而不得不花一些空间记忆指数n的变化过程（主要
是奇偶性变化）
     另外由于N的最大值不会超过(2^32 -1),所以数组个数取32 */
    int n,d;             /*辅助变量，d为数组l[]的游标 */
    d=0;

    for(n=N;n>=1;n=n/2){    /*计算逆向求解过程中，每一个中间指数的奇偶性，
并记录在数组中 */
        l[d]=2+n%2;         /*如果是偶数，则l[d]=2；如果是奇数，则
l[d]=3*/
```

```
        d++;
    }

    /*接下来根据l[d]中的记录开始正向计算result，d无需初始化，因为上一步得到的d值（为最大值）
    刚好就是所需要的 */
    for(;d>=0;d--){
        if (l[d]==3) result=result*result*X;   /*根据指数的奇偶性选择不同的公式 */
        else if (l[d]==2) result=result*result;

    }
    /*printf("%ld\n",result); /*用于检验结果*/
    return result;
}
```

Declaration

We hereby declare that all the work done in this project titled "Performance Measurement" is of our independent effort as a group.

Duty Assignments:

Programmer:张岱　Tester:毛顺龙　　Report Writer:陈潼