Performance Measurement (POW)

Date：2022-9-25

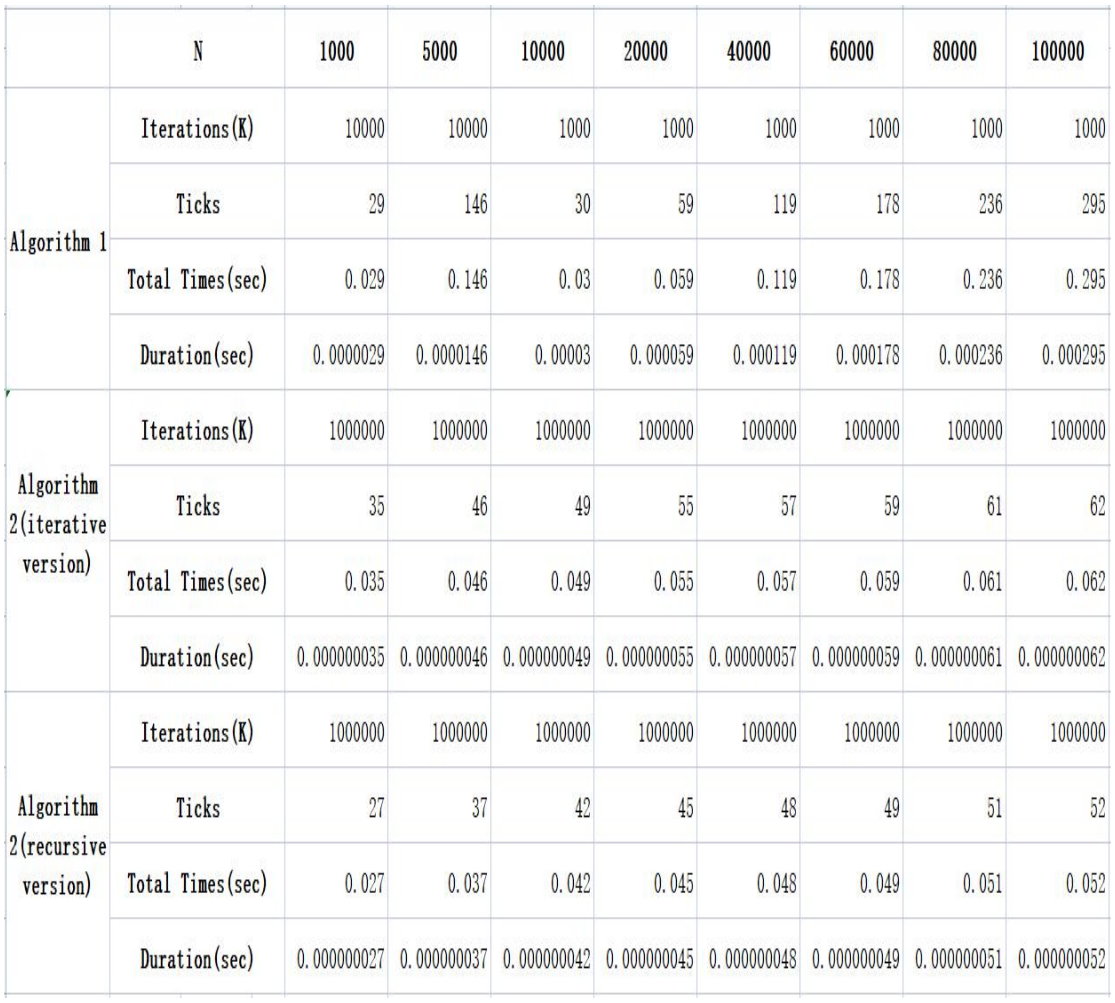
Chapter 1: Introduction

There are two different algorithms that can compute N power of X . Algorithm 1 is to use N−1 multiplications. Algorithm 2 works in the following way: if N is even, X(N)=X(N/2)×X(N/2); and if N is odd, X(N)=X((N−1)/2)×X((N−1)/2)×X, where X(N)means N power of X. We want to measure and compare the performances of Algorithm 1 and the iterative and recursive implementations of Algorithm 2 and analyze the complexities of the two algorithms.

Chapter 2: Algorithm Specification

Algorithm 1 works by using N-1 multiplications. Algorithm 2(iterative version) first determines the binary representation of N, uses an array to record the information of each bit, and then determines the parity according to each binary number, so as to obtain the N power of X according to the formula. Algorithm 2(recursive version) recursively obtains the Nth power of X according to the formula (if N is even, X(N)=X(N/2)×X(N/2); if N is odd, X(N)=X((N−1)/2)×X((N−1)/2)×X) .

Chapter 3:　Testing Results

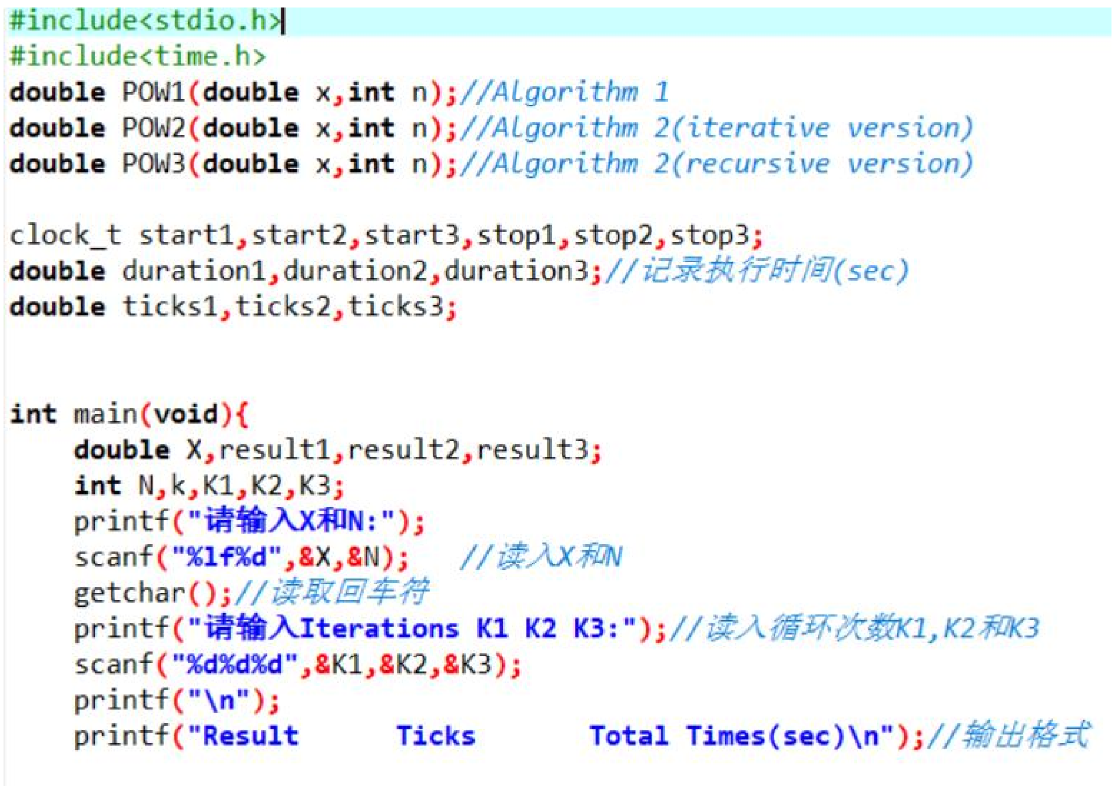


	N	1000	5000	10000	20000	40000	60000	80000	100000
Algorithm 1	Iterations(K)	10000	10000	1000	1000	1000	1000	1000	1000
	Ticks	29	146	30	59	119	178	236	295
	Total Times(sec)	0.029	0.146	0.03	0.059	0.119	0.178	0.236	0.295
	Duration(sec)	0.0000029	0.0000146	0.00003	0.000059	0.000119	0.000178	0.000236	0.000295
Algorithm 2(iterative version)	Iterations(K)	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000
	Ticks	35	46	49	55	57	59	61	62
	Total Times(sec)	0.035	0.046	0.049	0.055	0.057	0.059	0.061	0.062
	Duration(sec)	0.000000035	0.000000046	0.000000049	0.000000055	0.000000057	0.000000059	0.000000061	0.000000062
Algorithm 2(recursive version)	Iterations(K)	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000
	Ticks	27	37	42	45	48	49	51	52
	Total Times(sec)	0.027	0.037	0.042	0.045	0.048	0.049	0.051	0.052
	Duration(sec)	0.000000027	0.000000037	0.000000042	0.000000045	0.000000048	0.000000049	0.000000051	0.000000052

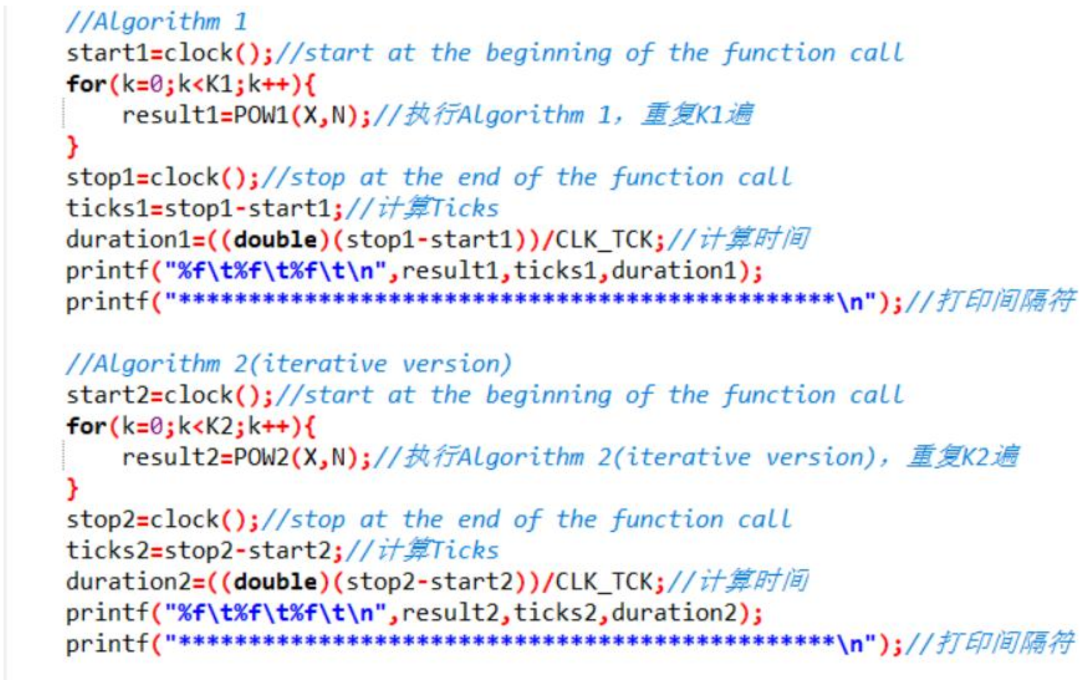
Chapter 4:　Analysis and Comments

For algorithm 1, the loop will execute N-1 times, so obviously its time complexity is O(N). For algorithm 2(both iterative and recursive version), the loop will execute log2(N) times at most, so obviously its time complexity is O(log N) .

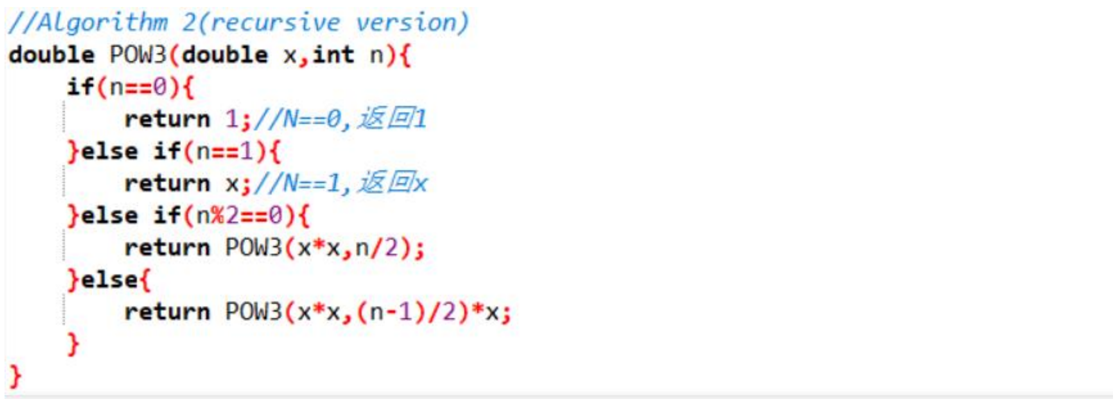
Appendix: Source Code (in C)



```c
#include<stdio.h>
#include<time.h>
double POW1(double x,int n);//Algorithm 1
double POW2(double x,int n);//Algorithm 2(iterative version)
double POW3(double x,int n);//Algorithm 2(recursive version)

clock_t start1,start2,start3,stop1,stop2,stop3;
double duration1,duration2,duration3;//记录执行时间(sec)
double ticks1,ticks2,ticks3;


int main(void){
    double X,result1,result2,result3;
    int N,k,K1,K2,K3;
    printf("请输入X和N:");
    scanf("%lf%d",&X,&N);   //读入X和N
    getchar();//读取回车符
    printf("请输入Iterations K1 K2 K3:");//读入循环次数K1,K2和K3
    scanf("%d%d%d",&K1,&K2,&K3);
    printf("\n");
    printf("Result      Ticks       Total Times(sec)\n");//输出格式
```



```c
    //Algorithm 1
    start1=clock();//start at the beginning of the function call
    for(k=0;k<K1;k++){
        result1=POW1(X,N);//执行Algorithm 1, 重复K1遍
    }
    stop1=clock();//stop at the end of the function call
    ticks1=stop1-start1;//计算Ticks
    duration1=((double)(stop1-start1))/CLK_TCK;//计算时间
    printf("%f\t%f\t%f\t\n",result1,ticks1,duration1);
    printf("************************************************\n");//打印间隔符

    //Algorithm 2(iterative version)
    start2=clock();//start at the beginning of the function call
    for(k=0;k<K2;k++){
        result2=POW2(X,N);//执行Algorithm 2(iterative version), 重复K2遍
    }
    stop2=clock();//stop at the end of the function call
    ticks2=stop2-start2;//计算Ticks
    duration2=((double)(stop2-start2))/CLK_TCK;//计算时间
    printf("%f\t%f\t%f\t\n",result2,ticks2,duration2);
    printf("************************************************\n");//打印间隔符
    //Algorithm 2(recursive version)
    start3=clock();//start at the beginning of the function call
    for(k=0;k<K3;k++){
        result3=POW3(X,N);//执行Algorithm 2(recursive version)，重复K3遍
    }
    stop3=clock();//stop at the end of the function call
    ticks3=stop3-start3;//计算Ticks
    duration3=((double)(stop3-start3))/CLK_TCK;//计算时间
    printf("%f\t%f\t%f\t\n",result3,ticks3,duration3);

    return 0;
}

//Algorithm 1
double POW1(double x,int n){
     int i;
     double y=1; //存放result
     for(i=0;i<n;i++){
         y=y*x;
     }
     return y;
}

//Algorithm 2(iterative version)
double POW2(double x,int n){
    int data[64],i,j;//data数组用于记录每次整除之后的奇偶性
    i=0;//初始化i，为后续循环遍历做准备
    double result=x;//存放结果
    if(n==0){
        return 1;//排除指数为0的情况;
    }
    while(n!=1){
        if(n%2==1){
            data[i]=1;//1:odd
            i++;
        }else{
            data[i]=0;
            i++;//0:even
        }
        n=n>>1;//n整除2
    }
    for(j=i-1;j>=0;j--){
        if(data[j]){
            result=result*result*x;//如果data[j]为1，说明此时指数为奇数
        }else{
            result=result*result;//如果data[j]为0，说明此时指数为偶数

        }
    }
    return result;
}
```



```
//Algorithm 2(recursive version)
double POW3(double x,int n){
    if(n==0){
        return 1;//N==0,返回1
    }else if(n==1){
        return x;//N==1,返回x
    }else if(n%2==0){
        return POW3(x*x,n/2);
    }else{
        return POW3(x*x,(n-1)/2)*x;
    }
}
```

Declaration

I hereby declare that all the work done in this project titled "Performance Measurement (POW)" is of my independent effort.