Performance Measurement of POW

Date: October-3th,2021

Chapter One: Introduction

In this project, we measure the performance of different algorithms, which are coded to compute X^N for some positive integer N.

As we all know, there are two ways usually used by programmers to achieve the goal. First way is computing directly, and we need use N−1 multiplications. Way two works as follow: 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.

Thus what we will do is as follow:

(1) Implement Algorithm 1 and an iterative version of Algorithm 2;

(2) Analyze the complexities of the two algorithms;

(3) Measure and compare the performances of Algorithm 1 and the iterative and recursive implementations of Algorithm 2 for X=1.0001 and N = 1000, 5000, 10000, 20000, 40000, 60000, 80000, 100000.

Chapter 2: Algorithm Specification

Description by pseudo-code and real code:

(1) Implement of algorithm one

```
Pow(int x,int n)
{
    Sum=1;                  //Initialize the output
    if n is zero            //analyse n is 0
    return 1;
    for i=1 to n{           //make a loop to compute n times
        Sum*=x;             // with i growing, sum change
    }
    return Sum;             //return final consequence
}
```

```
double Pow(double x,int i){
    double sum = 1;                         //Initialize the output
    if(i==0){                               //analyse n is 0
        return 1;                       //make a loop to compute n times
    }else{
        int j;
        for(j=1;j<=i;j++){                  // with j growing, sum change
            sum*=x;
        }
    }
    return sum;                             //output the consequence
}
```

(2) Iterative version of algorithm 2

```
Pow(int x,int n)
{
    sum = x;                //Initialize the output
    if n is zero            //analyse n is 0
```

```
        return 1;
    if n is 1;                    //analyse n is 1
        return x;
    Find a nearest but small number j to n,which is the
    power of 2;
    while(j>=1){              //essential loop
        if n/j is Exactly divisible by2
            x = x*x
        else
            x = x*x*sum;
    }
    return x;
}
```

```
double Pow(double x,int i){
    if(i==0){                                       //analyse when i is 0
        return 1;                                   //consequence is 1
    }
    if(i==1){                                       //ananlyse when i is 1
        return x;                                   //consequence is x
    }
    int j;                                          //auxiliary number
    double sum=x;
    for(j=2;j<=i;j=j*2);                            //calculate whether we should compute x extraly in certain loop
    j=j/4;                                          //find nearest 2 powers
    while(j>=1){
        if((i/j)%2==0){
            x = x*x;                                //when i/j is Exactly divisible by2
        }else{
            x = x*x*sum;                            //when i/j is not divisible by2
        }
        j=j/2;
    }
    return x;                                       //output value
}
```

(3) Recursive version of algorithm 2

```
Pow(int x,int n){
  if n==0                          //analyse special case 0
```

return 1

if n==1 //analyse special case 1

return x

if n%2==0

return Pow(x*x,n/2); //recursive

else

return Pow(x*x,n/2)*x;

```
double Pow(double x,int i){
    if(i == 0){
        return 1;                          //analyse when i is 0
    }
    if(i == 1){                            //ananlyse when i is 1
        return x;
    }
    if(i%2 == 0){                          //when i is divisible by 2
        return Pow(x*x,i/2);           //recursive
    }else{                                 //when i is not divisible by 2
        return Pow(x*x,i/2)*x;        //recursive
    }
}
```

Chapter 3: Testing Results

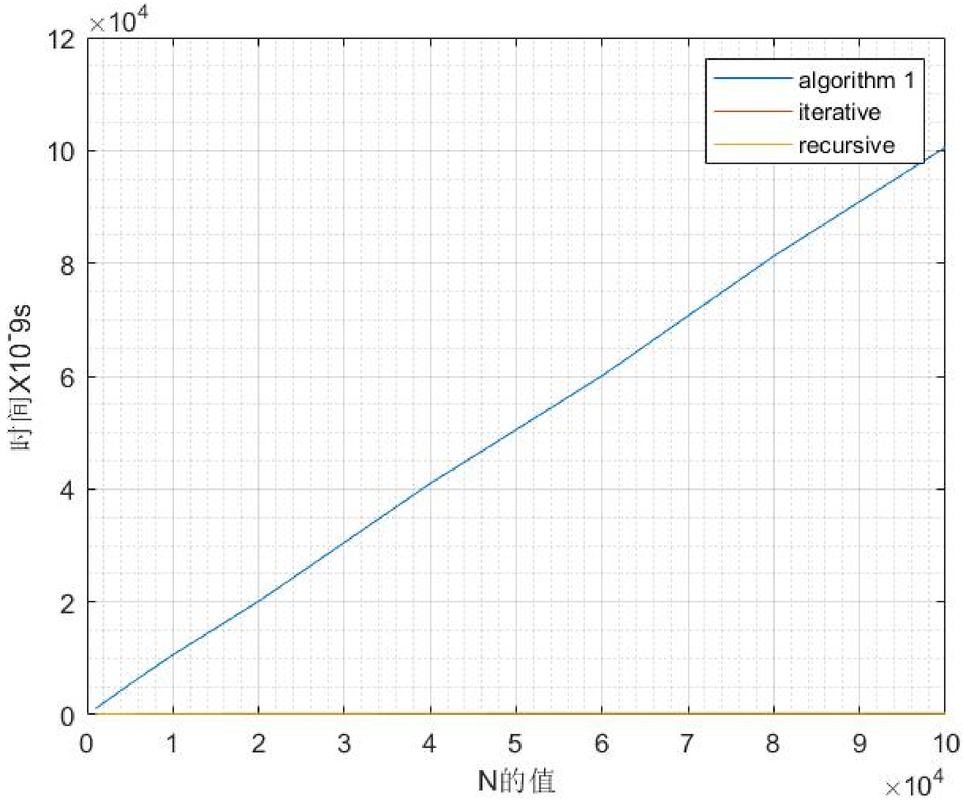
Be careful:

Because running time is really quickly, I decide to expand the number of runs to 10^4 times in algorithm 1 and 10^6 times in iterative algorithm and recursive algorithm. So all the consequence you read from the screen when you run .exe is 10000times or 10^6 times of the real duration

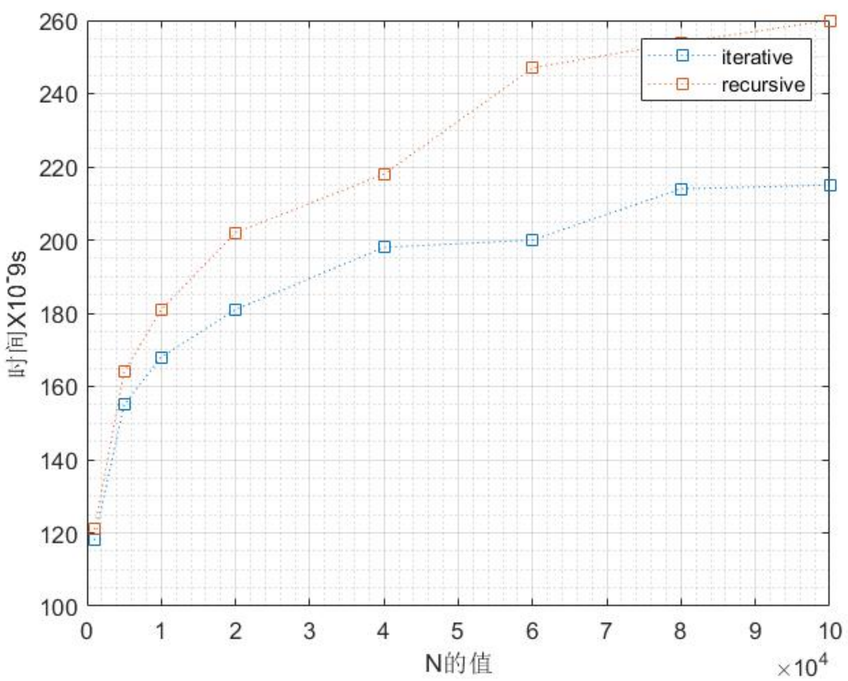
	N	1000	5000	10000	20000	40000	60000	80000	100000
Algorithm1	Iterations(K)	10^4							
	ticks	107	542	1006	2010	4100	6008	8130	10600
	Total time(s)	0.107	0.542	1.006	2.01	4.10	6.008	8.130	10.06
	Duration(x 10^{-4}s)	0.107	0.542	1.06	2.01	4.10	6.008	8.13	10.06
Algorithm 2(iteration)	Iterations(K)	10^6							
	ticks	118	155	168	181	198	200	214	215
	Total time(s)	0.118	0.155	0.168	0.181	0.198	0.2	0.214	0.215
	Duration(x 10^{-6}s)	0.118	0.155	0.168	0.181	0.198	0.2	0.214	0.215
Algorithm 2(recursive)	Iterations(K)	10^6							
	ticks	121	164	181	202	218	247	254	260
	Total time(s)	0.121	0.164	0.181	0.202	0.218	0.247	0.254	0.260
	Duration(x 10^{-6}s)	0.121	0.164	0.181	0.202	0.218	0.247	0.254	0.260

Draw a plot by matlab:

In this plot we can feel the time distinction between three algorithm directly.Apparently, algorithm 1 grows quicker than other extremely.



In this plot we can see the difference between iterative and recursive algorithm. Apparently, iterative is much better.



Chapter 4: Analysis and Comments

(1) Analyse and comments for the algorithm one:

a) Time Complexity:

In this algorithm, the time complexity is big O of n. Because to get the final number, we should compute one time with i grow to i+1.Therefore, if we input n,we should compute n time, which is defined as O(N).

b) Space Complexity:

In this algorithm, the space complexity is big O of 1. Because no matter what n and x we input, the space to store the value is a constant value.

c) Comments:

For this algorithm, the time complexity is big of n.It is so slow if n increases quickly.And this algorithm has advantage as well, the process is so clear that it is easy to understand.Besides, this algorithm take up little space.

(2) Analyse and comments for the algorithm two(iteration version):

a)Time Complexity:

In this algorithm, the time complexity is big O of log n. In this algorithm, to reach our goal, I have two main steps. In first step, we get the nearest but small power number of 2, which is O(log N),and in second step, we compute $\log_2 n$ times.

Thus ,the sum of O(log n) and O(log n) still is O(log n).

b)Space Complexity:

In this algorithm, the space complexity is the same as algorithm,which is big O of 1.No matter what n and x we input, the space to store the value is a constant value.

c)Comments:

For this algorithm, the time complexity is big of log n.It is so fast if n increases quickly.And this algorithm has other advantages as well, this algorithm take up little space.In a word, this is better than other two algorithm.

(3) Analyse and comments for the algorithm two(recursive version):

a)Time Complexity:

In this algorithm, the time complexity is big O of log n. It’s the same as iterative version.Every time when n is divisible by 2, we compute one time and call the next depth of recursive. Thus it’s O(log n).

b)Space Complexity:

In this algorithm, the space complexity is big O of log n.Every time when n is divisible by 2, we use extra certain space to store value.Thus it’ s O(log n).

c)Comments:

For this algorithm, the time complexity is big of log n.It is fast if n increases quickly.But the space complexity of the algorithm is big O of log n.With the number growing, the time it spend is growing as well.So this algorithm is not efficient.

Appendix: Source Code(in C)

Algorithm 1:

```c
#include<stdio.h>
#include<time.h>
clock_t start,stop;// Time recorder
double duration;// Time use in algorithm
double Pow(double x,int i);//Calculate function
int main(){
    double x,sum;                          //The number we will calculate
    int i,j;                               //The power rate is i
    printf("请输入x和i：\n");                //reminder to reader
    scanf("%lf %d",&x,&i);                 //input x and i
    start = clock();                       //Initialize the start time
    for(j=1;j<=10000;j++)                  //Expand 10000 times
    sum = Pow(x,i);                        //Calculate function
    printf("%lf\n",sum);                       //output calculate consequence
    stop = clock();                        //record final time
    duration = ((double) (stop - start))/CLK_TCK;//calculate duration
    printf("To increase run time,consider run 10000 times\n");
    printf("duration is %lf\nall time is %lf",duration,(double)(stop - start));//output value
}
double Pow(double x,int i){
    double sum = 1;                        //Initialize the output
    if(i==0){                              //analyse n is 0
        return 1;                        //make a loop to compute n times
    }else{
        int j;
        for(j=1;j<=i;j++){                 // with j growing, sum change
            sum*=x;
        }
    }
    return sum;                            //output the consequence
}
```

Algorithm 2(iterative)

```c
#include<stdio.h>
#include<time.h>
clock_t start,stop;// Time recorder
double duration;// Time use in algorithm
double Pow(double x,int i);//Calculate function
int main(){
    double x,sum;                              //The number we will calculate
    int i,j;                                   //The power rate is i
    printf("请输入x和i：\n");                 //reminder to reader
    scanf("%lf %d",&x,&i);                    //input x and i
    start = clock();                          //Initialize the start time
    for(j=1;j<=1000000;j++)                   //Expand 1000000 times
    sum = Pow(x,i);                           //Calculate function
    printf("%lf\n",sum);                       //output calculate consequence
    stop = clock();                           //record final time
    duration = ((double) (stop - start))/CLK_TCK;//calculate duration
    printf("To increase run time,consider run 1000000 times\n");
    printf("duration is %lf\nall time is %lf",duration,(double)(stop - start));//output value
}
double Pow(double x,int i){
    if(i==0){                                 //analyse when i is 0
        return 1;                             //consequence is 1
    }
    if(i==1){                                 //ananlyse when i is 1
        return x;                             //consequence is x
    }
    int j;                                    //auxiliary number
    double sum=x;
    for(j=2;j<=i;j=j*2);                      //calculate whether we should compute x extraly in certain loop
    j=j/4;                                    //find nearest 2 powers
    while(j>=1){
        if((i/j)%2==0){
            x = x*x;                          //when i/j is Exactly divisible by2
        }else{
            x = x*x*sum;                      //when i/j is not divisible by2
        }
        j=j/2;
    }
    return x;                                 //output value
}
```

Algorithm 2(recursive)

```c
#include<stdio.h>
#include<time.h>
clock_t start,stop;// Time recorder
double duration;// Time use in algorithm
double Pow(double x,int i);//Calculate function
int main(){
    double x,sum;                              //The number we will calculate
    int i,j,k;                                 //The power rate is i
    printf("请输入x和i: \n");                  //reminder to reader
    scanf("%lf %d",&x,&i);                     //input x and i
    start = clock();                           //Initialize the start time
    for(j=1;j<=1000000;j++)                    //Expand 1000000 times
    sum = Pow(x,i);                            //Calculate function
    printf("%lf\n",sum);                       //output calculate consequence
    stop = clock();                            //record final time
    duration = ((double) (stop - start))/CLK_TCK;//calculate duration
    printf("To increase run time,consider run 1000000 times\n with i = %d",i);
    printf("duration is %lf\nall time is %lf\n",duration,(double)(stop - start));//output value
}
double Pow(double x,int i){
    if(i == 0){
        return 1;                              //analyse when i is 0
    }
    if(i == 1){                                //ananlyse when i is 1
        return x;
    }
    if(i%2 == 0){                              //when i is divisible by 2
        return Pow(x*x,i/2);            //recursive
    }else{                                     //when i is not divisible by 2
        return Pow(x*x,i/2)*x;         //recursive
    }
}
```

Be careful:All algorithm above can be found in the compressed file.

Declaration

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