Report of Performance Measurement

Author Names

September 24, 2019

Chapter 1

Introduction

To solve the problem,we have two assignments :

1.Implement three function

a.use $N-1$ multiplications to compute X^N

b.compute X^N recursively

c.compute X^N iteratively

2.Measure and compare the performances of the three function

The problem requires us to use the situation of $X = 1.0001$ and $N = 1000, 5000, 10000, 20000, 40000, 60000, 80000, 100000$.

Chapter 2

Algorithm Specification

Here are pseudo-codes for three functions:

2.1 Function 1 (multiplications)

Input: radix x , exponent N Output: x^N

1 Use algorithms 1 to compute x^N

```
function FUNCTION1(x, N)
    result ← 1
    for i = 0 → N do
        result ← result * x
    end for
    return result
end function
```

2.2 Function 2 (recursion)

Input: radix x , exponent N Output: x^N

2 Use algorithms 2 to compute x^N recursively

```
function FUNCTION2RECUR(x, N)
    if N = 0 then
        return 1
    end if
    if N = 1 then
        return x
    end if
    if N%2 = 1 then
        return function2recur(x ∗ x, N/2) ∗ x
    else
        return function2recur(x ∗ x, N/2)
    end if
end function
```

2.3 Function 3 (iteration)

Input: radix x , exponent N Output: x^N

3 Use algorithms 2 to compute x^N recursively

```
function FUNCTION2RECUR(x, N)
    result ← 1
    while i ≠ 0 do
        if i mod 2 ≠ 0 then
            result ← result ∗ x
        end if
        x ← x ∗ x
    end while
    return result
end function
```

Chapter 3

Testing Results

purpose : to test the performance of each algorithm when N is relevantly small (to check the coefficient before O)

expected result : since algorithm1 has O(n) time complexity,and does not have prominent smaller constant coefficient than algorithm 2(recursive and iterative form),it is expected to run much slower than algorithm 2 actual behavior meet the expectation

current status : All passed

Table 3.1

	N	1000			
1	iterations(K)	1.00E+05	1.00E+05	1.00E+05	
	ticks(sec)	306.00000	309.00000	310.00000	
	Total Time(sec)	0.30600	0.30900	0.31000	
	Duration(sec)	3.06E-06	3.09E-06	3.10E-06	3.08E-06
2	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	128.00000	129.00000	127.00000	
	Total Time(sec)	0.12800	0.12900	0.12700	
	Duration(sec)	2.56E-08	2.58E-08	2.54E-08	2.56E-08
3	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	188.00000	189.00000	188.00000	
	Total Time(sec)	0.18800	0.18900	0.18800	
	Duration(sec)	3.76E-08	3.78E-08	3.76E-08	3.77E-08
	N	5000			
1	iterations(K)	5.00E+03	5.00E+03	5.00E+03	
	ticks(sec)	76.00000	76.00000	77.00000	
	Total Time(sec)	0.07600	0.07600	0.07700	
	Duration(sec)	1.52E-05	1.52E-05	1.54E-05	1.53E-05
2	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	168.00000	164.00000	171.00000	
	Total Time(sec)	0.16800	0.16400	0.17100	
	Duration(sec)	3.36E-08	3.28E-08	3.42E-08	3.35E-08
3	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	247.00000	251.00000	262.00000	
	Total Time(sec)	0.24700	0.25100	0.26200	
	Duration(sec)	4.94E-08	5.02E-08	5.24E-08	5.07E-08
	N	10000			
1	iterations(K)	5.00E+03	5.00E+03	5.00E+03	
	ticks(sec)	157.00000	160.00000	155.00000	
	Total Time(sec)	0.15700	0.16000	0.15500	
	Duration(sec)	3.14E-05	3.20E-05	3.10E-05	3.15E-05
2	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	173.00000	176.00000	176.00000	
	Total Time(sec)	0.17300	0.17600	0.17600	
	Duration(sec)	3.46E-08	3.52E-08	3.52E-08	3.50E-08
3	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	272.00000	273.00000	274.00000	
	Total Time(sec)	0.27200	0.27300	0.27400	
	Duration(sec)	5.44E-08	5.46E-08	5.48E-08	5.46E-08

Table 3.2

	N	20000			
1	iterations(K)	5.00E+03	5.00E+03	5.00E+03	
	ticks(sec)	318.00000	318.00000	311.00000	
	Total Time(sec)	0.31800	0.31800	0.31100	
	Duration(sec)	6.36E-05	6.36E-05	6.22E-05	6.31E-05
2	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	194.00000	198.00000	186.00000	
	Total Time(sec)	0.19400	0.19800	0.18600	
	Duration(sec)	3.88E-08	3.96E-08	3.72E-08	3.85E-08
3	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	309.00000	295.00000	300.00000	
	Total Time(sec)	0.30900	0.29500	0.30000	
	Duration(sec)	6.18E-08	5.90E-08	6.00E-08	6.03E-08
	N	40000			
1	iterations(K)	5.00E+03	5.00E+03	5.00E+03	
	ticks(sec)	620.00000	624.00000	620.00000	
	Total Time(sec)	0.62000	0.62400	0.62000	
	Duration(sec)	1.24E-04	1.25E-04	1.24E-04	1.24E-04
2	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	202.00000	201.00000	197.00000	
	Total Time(sec)	0.20200	0.20100	0.19700	
	Duration(sec)	4.04E-08	4.02E-08	3.94E-08	4.00E-08
3	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	322.00000	323.00000	322.00000	
	Total Time(sec)	0.32200	0.32300	0.32200	
	Duration(sec)	6.44E-08	6.46E-08	6.44E-08	6.45E-08
	N	60000			
1	iterations(K)	5.00E+03	5.00E+03	5.00E+03	
	ticks(sec)	931.00000	926.00000	928.00000	
	Total Time(sec)	0.93100	0.92600	0.92800	
	Duration(sec)	1.86E-04	1.85E-04	1.86E-04	1.86E-04
2	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	203.00000	203.00000	206.00000	
	Total Time(sec)	0.20300	0.20300	0.20600	
	Duration(sec)	4.06E-08	4.06E-08	4.12E-08	4.08E-08
3	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	330.00000	330.00000	334.00000	
	Total Time(sec)	0.33000	0.33000	0.33400	
	Duration(sec)	6.60E-08	6.60E-08	6.68E-08	6.63E-08

Table 3.3

	N	80000			
1	iterations(K)	5.00E+03	5.00E+03	5.00E+03	
	ticks(sec)	1239.00000	1243.00000	1242.00000	
	Total Time(sec)	1.23900	1.24300	1.24200	
	Duration(sec)	2.48E-04	2.49E-04	2.48E-04	2.48E-04
2	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	209.00000	208.00000	210.00000	
	Total Time(sec)	0.20900	0.20800	0.21000	
	Duration(sec)	4.18E-08	4.16E-08	4.20E-08	4.18E-08
3	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	343.00000	340.00000	342.00000	
	Total Time(sec)	0.34300	0.34000	0.34200	
	Duration(sec)	6.86E-08	6.80E-08	6.84E-08	6.83E-08
	N	100000			
1	iterations(K)	5.00E+03	5.00E+03	5.00E+03	
	ticks(sec)	1591.00000	1567.00	1577	
	Total Time(sec)	1.59100	1.56700	1.57700	
	Duration(sec)	3.18E-04	3.13E-04	3.15E-04	3.16E-04
2	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	211.00000	217.00	2.11E+02	
	Total Time(sec)	0.21100	0.21700	0.21100	
	Duration(sec)	4.22E-08	4.34E-08	4.22E-08	4.26E-08
3	iterations(K)	5.00E+06	5.00E+06	5.00E+06	
	ticks(sec)	357.00000	354.00	3.51E+02	
	Total Time(sec)	0.35700	0.35400	0.35100	
	Duration(sec)	7.14E-08	7.08E-08	7.02E-08	7.08E-08

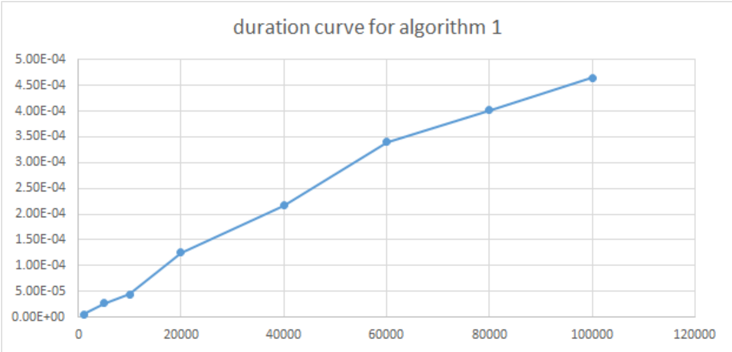

Figure 3.1: Duration curve for algorithm1

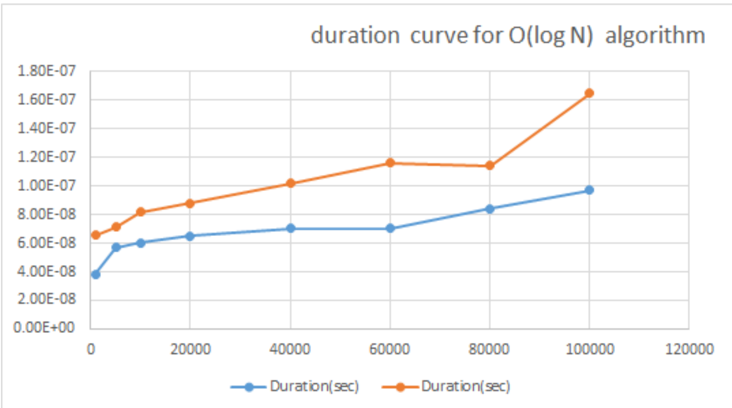

Figure 3.2: Duration curve for $(OlogN)$ algorithm

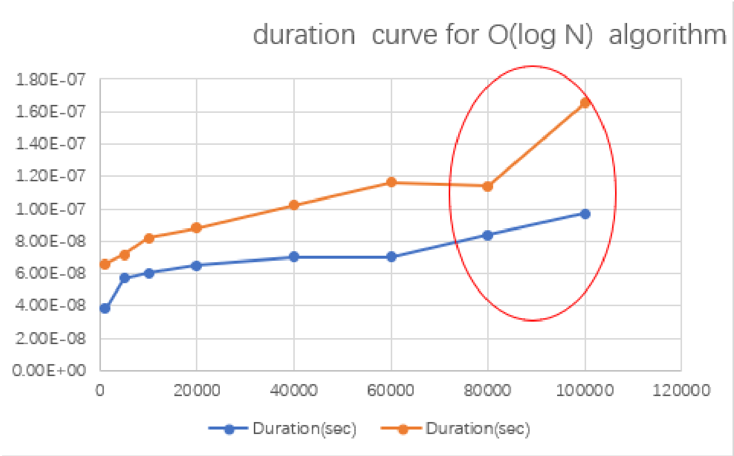

Figure 3.3: Duration curve for $(OlogN)$ algorithm

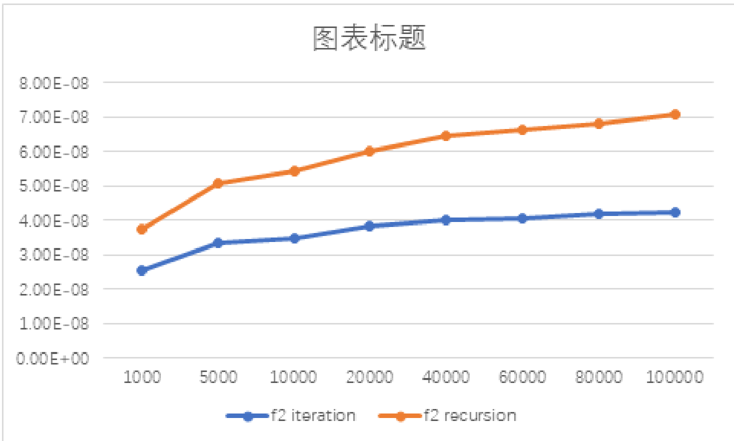

Figure 3.4: Duration curve for $(OlogN)$ algorithm

Figure3.3 shows the first testing result. The orange curve represent recursion and blue one represents iteration. We can see that when$N = 60000$, the average running time is abnormally lower than$N = 80000$. This is mainly due to fluctuations. And in the first test, we did not consider repetition, that is the main source of errors and residuals. Another reason which might result in errors and residuals is in iteration.

```
double f2_iter(double x,int N){
        double result = 1;
        for(int i = N;i!=0;i/=2){
                if(i%2 !=0){
                        result *=x;
                }
                else result=result*1;
                x*=x;
        }
        return result;
}
```

In the first test version we do not have the line painted red, this line actually does nothing but to compensate for the running time. Consider how our algorithm(iteration) works. The line $result = result * x$ runs only when the binary position is 1, and if the binary position is 0, we'll miss one instruction (comparing to 1).

Let's take $N = 60000$ and$N = 80000$ as examples. binary form of 60000 and 80000, Consider their binary forms $60000 = 1110101001100000$(The number of 1 is 7) and $80000 = 10011100010000000$(The number of 1 is 5).By estimation if we do not add the red line, the difference between $N = 60000$ and$N = 80000$would narrow since when we test $N = 60000$ the $result = result * x$would run more times than $N = 80000$, which causes errors and residuals.

In the second version(Figure 3.4) of our test we test 3 times for each N, calculate the average duration to reduce the errors, and we add the red line to make sure the ensure the result.

Chapter 4

Analysis and Comments

Table 4.1: Add caption

	Algorithm1	Algorithm2(iterative)	Algorithm3(recursive)
Time complexity	O(N)	$O(logN)$	$O(logN)$
Space complexity	O(1)	$O(1)$	$O(logN)$

1. algorithm 1 this algorithm is simply brute force.

2. algorithm 2 recursive form: we list recursive form before the iterative because the former takes more space. For each time we reduce the power by half (for example 16-8), we have to judge 1-3 times, and then enter a function. The function requires space in the stack, and this space won't be freed unless we reached N=0 or N=1; so the time complexity and space complexity are both $O(logN)$

Stack(take $N = 16$ as example):

Please refer to next page.

Table 4.2: STACK

TOP_OF_STACK
Data x, N(N==0)
Pointer to former function3
Function4
.
Data x ,N (N=8)
Pointer to Function1
Function2
Data x, N(N=16)
Pointer to f2
Function1

3. algorithm2 iterative form the hardest point about this question is that we have to judge whether the power is even or odd. In the iterative form the power starts from max to min (e.g. 61->30->15->...) , while the exponent for x starts from min to max (1->3->6->13), and its impossible to tell whether the exponent for x apply for the odd branch or even branch when the power is yet in the higher level.

The simple solution is to build an array to record the whether the power is odd or even, but this takes $O(logN)$ space. Next we introduce a new method which takes $O(1)$ space complexity and $O(logN)$ time complexity. Notice we want to calculate x^N , if we know Ncan be written as$N = N_1 + N_2 + ... + N_k$ Then $x^N = x_1^N * x_2^N * ... * x_k^N$, if k is$O(logN)$, and we manage to getN_2 from $N_1, N_3 from N_2, ..N_k from N_{k-1}$, this will not require $O(logN)$ extra space. And we can link the partition of a natural number with the binary expression.

To make clear of how the algorithm works, we take$N = 46$ as example. $46D = 101110B$, notice we get the 0 marked red first. We choose the coefficient between 0 and 1, by calculating $N \bmod 2$. If the coefficient is 1,we add current x to result. After this we let $x = x * x$, and do the next iteration. In this way we do not use extra space, and the time complexity remains$O(logN)$.

Table 4.3: Add caption

Times for calculation	X_cur	Coefficient	Result
1	2^0	0	
2	2^1	1	2
3	2^2	1	$2+4$
4	2^3	1	$2+4+8$
...	...	...	...

Chapter 5

Appendix: Source Code

```c
#include <stdio.h>
#include <time.h>


clock_t start,stop;
double duration1,duration2,duration3;
//define duration
double toltime1,toltime2,toltime3;
//define toltime

//define Iterations
#define MAXK1 5e3
#define MAXK2 5e6
#define MAXK3 5e6

double f1(double x,int N);
//this function uses Algorithm 1
double f2_iter(double x,int N);
//this function uses iterative version of Algorithm 2
double f2_recur(double x,int N);
//this function uses recursive version of Algorithm 2

int main(){
```

```
int N;
double x;
double result1,result2,result3;
printf("Please input base: ");
//explain of input
scanf("%lf",&x);
printf("Please input exponent: ");
//explain of input
scanf("%d",&N);
start= clock();
//time initialize
for(int i=0;i<MAXK1;i++)
// run MAXK1 times
result1 = f1(x,N);
//call function f1
stop = clock();
duration1 = ((double)(stop - start))/CLK_TCK/MAXK1;
//compute durations
toltime1 =  ((double)(stop - start))/CLK_TCK;
printf("Algorithm 1 : \n");
// output initialize
printf(" Iterations: %.0e\n",MAXK1);
// ouput iterations
printf(" ticks1 = %f\n",(double)(stop - start));
printf(" tolal time = %5lf\n",toltime1);
//output toltime
printf(" duration1 = %6.2e\n",duration1);
start= clock();
//time initialize
for(int i=0;i<MAXK2;i++)
// run MAXK2 times
        result2 = f2_iter(x,N);
        //call function f2
```

```
        stop = clock();
        duration2 = ((double)(stop - start))/CLK_TCK/MAXK2;
        //compute durations
        toltime2 =  ((double)(stop - start))/CLK_TCK;
        printf("Algorithm 2,iterative form : \n");
        // output initialize
        printf(" Iterations: %.0e\n",MAXK2);
        // ouput iterations
        printf(" ticks2 =%f\n",(double)(stop - start));
        printf(" tolal time = %5lf\n",toltime2);
        //output toltime
        printf(" duration2 = %6.2e\n",duration2);

        start= clock();//time initialize
        for(int i=0;i<MAXK3;i++)
                result3 = f2_recur(x,N);
        stop = clock();
        duration3 = ((double)(stop - start))/CLK_TCK/MAXK3;
        toltime3 =  ((double)(stop - start))/CLK_TCK;
        printf("Algorithm 2,recursive form : \n");
        // output initialize
        printf(" Iterations: %.0e\n",MAXK3);
        printf(" ticks3 = %f\n",(double)(stop - start));
        printf(" tolal time = %5lf\n",toltime3);
        printf(" duration3 =%6.2e\n",duration3);
        return 0;
}

double f1(double x,int N){
          double result=1;2
        for(int i=1;i<=N;i++){
                result = result * x;
        }
```

```
        return result;
}


double f2_iter(double x,int N){
        double result = 1;
        for(int i = N;i!=0;i/=2){
        //iterative
                if(i%2 !=0){
                        result *=x;
                }
                x*=x;
        }
        return result;
}

double f2_recur(double x,int N){
        //recursive
        if(N==0)
                return 1;
        if(N==1)
                return x;
        if(N%2==1)
                return f2_recur(x*x,N/2)*x;
        else
                return f2_recur(x*x,N/2);
}
```

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: xxx

Tester: xxx

Report Writer: xxx