Fundamentals of Data Structures

Laboratory Projects

PROJECT 1: PERFORMANCE MEASUREMENT (POW)

Author's Name

DATE: 2023-10-9

CHAPTER 1: INTRODUCTION

In this report, we will show and analysis two different algorithms that can be used to compute X^N for some positive integer N.

Algorithm 1 is to simply use $N-1$ multiplications.

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

When Algorithm 2 is uesd, we should try both **recursive** version and **iterative** version.

Both algorithms will be tested when $X = 1.0001$ and N = 1000, 5000, 10000, 20000, 40000, 60000, 80000,100000

CHAPTER 2: ALGORITHM SPECIFICATION

Main Program

In the main program, we set clocks and the callings of the functions of algorithms and compute directly and compare the answer with each other at first.The test part are mainly combined with three similar parts for three versions of algorithms.And we also need to set the value of K .

When we are testing, we need to count the time of the function.

The outputs are also contained in the main part. It will print out total time and durations of each versions of algorithms.

Algorithm 1

This algorithm uses loops to multiply the result by x for N times.

```
double pow1(double X,int N)
{
    double res=1;
    while(N--) res*=X;//use for-loop to multiply X repetitively
    return res;
}
```

Algorithm 2 (recursive version)

Algorithm2 uses recursion to call the value of $X^{N/2}$ (or $X^{(N-1)/2}$). In this way, we can save a lot of time because some repeated calculation could be reused.

In this algorithm, we use `if` statement to judge parity of the given N and provide an exit of the recursion.

```
double pow2_rec(double X,int N)
{
    if (N == 0) return 1;
    if (N == 1) return X;
    if (N % 2 == 0) return pow2_rec(X * X, N / 2); // return the result when n is even
    else return pow2_rec(X * X, N / 2) * X; // multiply extra x when n is odd
}
```

Algorithm 2 (iterative version)

In this version of the Algorithm, we use the `while` loop to replace the recursion ahead. In each loop, it will judge the parity of N (if n is odd, multiply an x exactly), and do preparations for the next loop.

```
double pow2_ite(double X,int N)
{
    double result = 1;
    while (N > 0){
        if (N % 2 == 1) result = result * X; /*multiply extra x when n is odd; store calculation
        to result when n = 1*/
        X = X * X;
        N = N / 2; //get prepared of x^(n/2) * x^(n/2) when n is even
    }
    return result;
}
```

CHAPTER 3: TESTING RESULTS

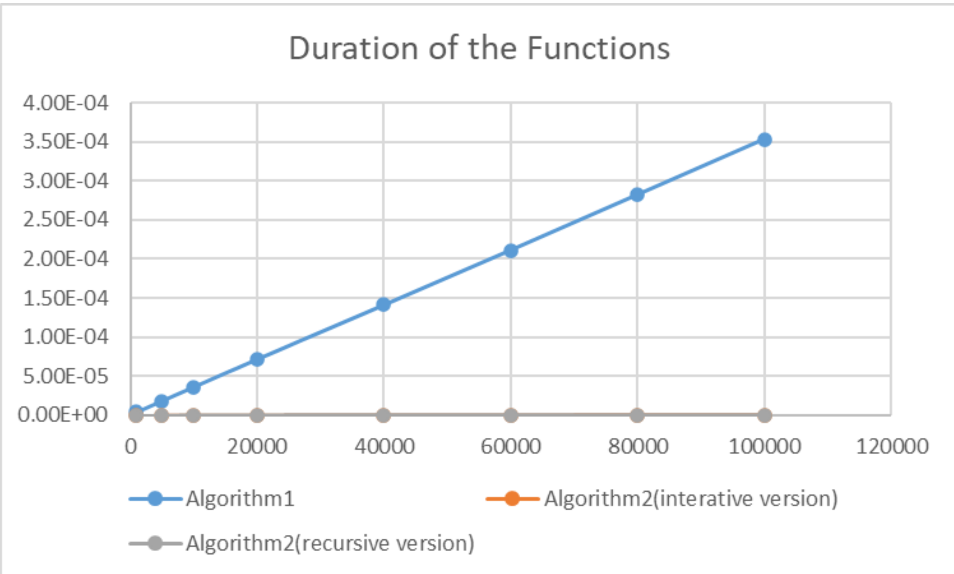
Test Data Sheet

```
exponent                                 1000          5000          10000         20000         40000         60000          80000          100000


Algorithm1                              1.105165      1.648680      2.718146      7.388317      54.587232     403.307791     2979.765922    22015.456049
Algorithm2(interative version)          1.105165      1.648680      2.718146      7.388317      54.587232     403.307791     2979.765922    22015.456048
Algorithm2(recursive version)           1.105165      1.648680      2.718146      7.388317      54.587232     403.307791     2979.765922    22015.456048
                                                                    Algorithm1
K:                                      10000         10000         10000         10000         10000         10000          10000          10000
Duration(e-4):                          0.046000      0.169000      0.242000      0.478000      0.961000      1.425000       1.922000       2.386000
Total Time:                             0.046000      0.169000      0.242000      0.478000      0.961000      1.425000       1.922000       2.386000
                                                                    Algorithm2(interative version)
K:                                      10000000      10000000      10000000      10000000      10000000      10000000       10000000       10000000
Duration(e-7):                          0.233000      0.303000      0.371000      0.352000      0.401000      0.447000       0.412000       0.427000
Total Time:                             0.233000      0.303000      0.371000      0.352000      0.401000      0.447000       0.412000       0.427000
                                                                    Algorithm2(interative version)
K:                                      10000000      10000000      10000000      10000000      10000000      10000000       10000000       10000000
Duration(e-7):                          0.157000      0.178000      0.224000      0.247000      0.262000      0.259000       0.267000       0.282000
Total Time:                             0.157000      0.178000      0.224000      0.247000      0.262000      0.259000       0.267000       0.282000
```

N		1000	5000	10000	20000	40000	60000	80000	100000
Algorithm1	Iterations（K)	1.00E+04	1.00E+04	1.00E+04	1.00E+04	1.00E+04	1.00E+04	1.00E+04	1.00E+04
	Ticks	37	181	359	713	1411	2111	2817	3530
	Total Time（sec)	0.037	0.181	0.359	0.713	1.411	2.111	2.817	3.530
	Duration(sec)	3.70E-06	1.81E-05	3.59E-05	7.13E-05	1.41E-04	2.11E-04	2.82E-04	3.53E-04
Algorithm2(interative version)	Iterations（K)	1.00E+07	1.00E+07	1.00E+07	1.00E+07	1.00E+07	1.00E+07	1.00E+07	1.00E+07
	Ticks	294	376	416	440	461	472	573	579
	Total Time（sec)	0.294	0.376	0.416	0.440	0.461	0.472	0.573	0.579
	Duration(sec)	2.99E-08	3.76E-08	4.16E-08	4.40E-08	4.61E-08	4.72E-08	5.73E-08	5.79E-08
Algorithm2(recursive version)	Iterations（K)	1.00E+07	1.00E+07	1.00E+07	1.00E+07	1.00E+07	1.00E+07	1.00E+07	1.00E+07
	Ticks	455	604	685	734	803	830	842	843
	Total Time（sec)	0.455	0.604	0.685	0.734	0.803	0.830	0.842	0.843
	Duration(sec)	4.55e10-8	6.04E-08	6.85E-08	7.34E-08	8.03E-08	8.30E-08	8.42E-08	8.43E-08

Graph



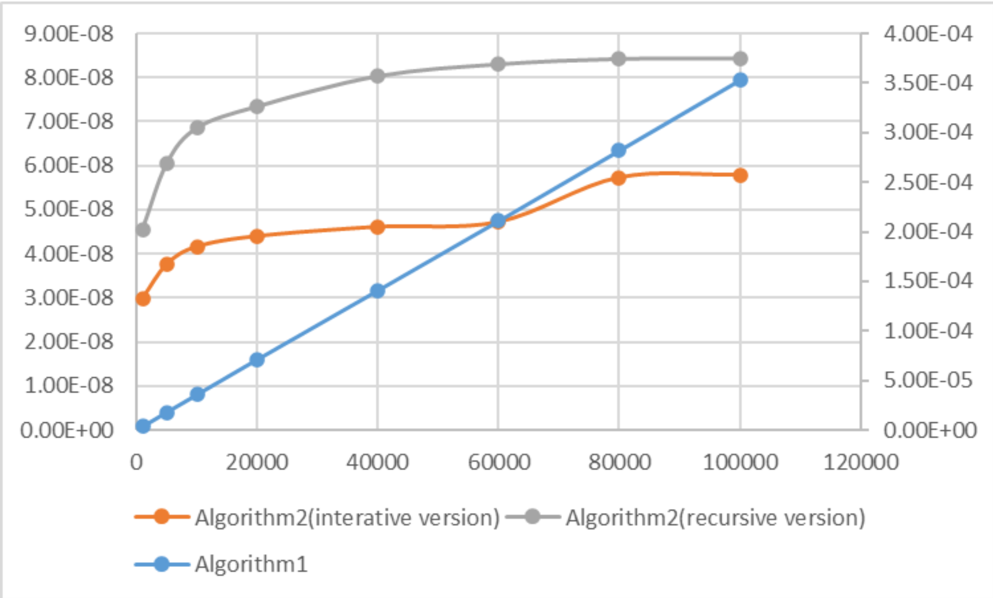
In the graph, we can easily find that the duration of algorithm1 is much bigger than algorithm2.

And as N grows larger, the gap between different algorithms grows larger.

We can know from the trend in the graph that the time complexity of algorithm1 is O(N).

However, in this graph, we can't see the trend of the two versions of algorithm2 clearly because the value of duration is too small to tell.

So another plotyy is showed as below:



In this one, we can see the similar trend of algorithm2,no matter which version. We can judge that their time complexity are both O(logN) roughly.

We can also see that iterative version has better performance than recursive version. But their order of magnitudes are the same.

CHAPTER 4: ANALYSIS AND COMMENTS

Algorithm 1

In algorithm 1, we use ***for*** loop to multiply X for N times. As the ***for*** loop will run N times, the time complexity of this algorithm is O(N)

Meanwhile, we can see from the code that it needn't extra space that is related to N, so the space complexity of algorithm is O(1).

Comprehensively,this algorithm can't use the computation sufficiently. for each X^N , it multiplies X on the result one by one. So it is not a Efficient algorithm.

Algorithm 2 (recursive version)

In Algorithm 2 (recursive version), each calculation is divided into two parts, which will stop when the value of N turns zero. So the time complexity is O(logN)

The space complexity of this algorithm is O(logN) because it uses as the same space as the number of recursion.

This algorithm is can use the result of $X^{N/2}$ (or $X^{(N-1)/2}$) to reduce some repeated computation. So it seems a better algorithm.

Algorithm 2 (iterative version)

The main idea of this version is as the same as the recursion version.

So the time complexity is O(logN)).

But it uses the ***while*** loop to realize it. Compared to the recursion, the ***while*** loop don't need to call the function again, so it is faster than recursion.

The space complexity of this algorithm is O(1) because it do not need space related to N.

APPENDIX:SOURCE CODE (IN C)

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

clock_t start,stop; //clock_t is a built-in type for processor time (ticks)
clock_t begin,end; //clock_t is a built-in type for processor time (ticks)
double duration,que[9]; /*records the run time (seconds) of a function and que is Used
to store Total Time for each N */

double pow1(double X,int N)
{
    double res=1;
    while(N--) res*=X;//use for-loop to multiply X repetitively
    return res;
}
double pow2_ite(double X,int N)
{
    double result = 1;
    while (N > 0){
```

```
        if (N % 2 == 1) result = result * X; /*multiply extra x when n is odd; store
calculation
        to result when n = 1*/
        X = X * X;
        N = N / 2; //get prepared of x^(n/2) * x^(n/2) when n is even
    }
    return result;
}
double pow2_rec(double X,int N)
{
    if (N == 0) return 1;
    if (N == 1) return X;
    if (N % 2 == 0) return pow2_rec(X * X, N / 2); // return the result when n is even
    else return pow2_rec(X * X, N / 2) * X; // multiply extra x when n is odd
}

int main(){
    double X=1.0001; //set test base number
    int N[8]={1e3,5e3,1e4,2e4,4e4,6e4,8e4,1e5}; //set an array to store N
    printf("exponent                        ");
    for(int i=0;i<8;i++) printf("%16d",N[i]);
    printf("\n\t\t\t\t\t\t\t\t\t\t\t\t\t  \n");
    //printf("\t��������");
    printf("\nAlgorithm1                        ");
    for(int i=0;i<8;i++)  printf("%16lf",pow1(X,N[i])); //output result
    printf("\nAlgorithm2(interative version)  ");
    for(int i=0;i<8;i++)  printf("%16lf",pow2_ite(X,N[i])); //output result
    printf("\nAlgorithm2(recursive version)   ");
    for(int i=0;i<8;i++)  printf("%16lf",pow2_rec(X,N[i])); //output result

    int K=1e4;//K stands for the number of times this function runs repeatedly
    printf("\n\t\t\t\t\t\t\t\t\t\tAlgorithm1  ");
    printf("\nK:                              ");
    for(int i=0;i<8;i++) printf("%16d",K);
    printf("\nDuration(e-4):                  ");
    for(int i=0;i<8;i++){
        K=1e4;
        begin=clock(); //records the ticks at the beginning of the function call
        start = clock();
        while(K--) pow1(X,N[i]); //run function
        stop = clock(); //records the ticks at the end of the function call
        duration =((double)(stop-start))/CLK_TCK;//calculate duration for a single run of the
function
        printf("%16lf",duration); //output duration
        end=clock();
        que[i]=((double)(end-begin))/CLK_TCK; //calculate Total Time for a repetition
    }
    printf("\nTotal Time:                     ");
    for(int i=0;i<8;i++){
        printf("%16lf",que[i]); //output each Total Time
    }

    K=1e7; //K stands for the number of times this function runs repeatedly
    printf("\n\t\t\t\t\t\t\t\t\t\tAlgorithm2(interative version)  ");
    printf("\nK:                              ");
```

```
    for(int i=0;i<8;i++) printf("%16d",K);
    printf("\nDuration(e-7):                          ");
    for(int i=0;i<8;i++){
        K=1e7;
        begin=clock(); //records the ticks at the beginning of the function call
        start = clock();
        while(K--) pow2_rec(X,N[i]);//run function
        stop = clock();//records the ticks at the end of the function call
        duration =((double)(stop-start))/CLK_TCK;//calculate duration for a single run of the
function
        printf("%16lf",duration);//output duration
        end=clock();
        que[i]=((double)(end-begin))/CLK_TCK;//calculate Total Time for a repetition
    }
    printf("\nTotal Time:                             ");
    for(int i=0;i<8;i++){
        printf("%16lf",que[i]);//output each Total Time
    }

    K=1e7; //K stands for the number of times this function runs repeatedly
    printf("\n\t\t\t\t\t\t\t\t\t\tAlgorithm2(interative version)  ");
    printf("\nK:                                      ");
    for(int i=0;i<8;i++) printf("%16d",K);
    printf("\nDuration(e-7):                          ");
    for(int i=0;i<8;i++){
        K=1e7;
        begin=clock(); //records the ticks at the beginning of the function call
        start = clock();
        while(K--) pow2_ite(X,N[i]);//run function
        stop = clock();//records the ticks at the end of the function call
        duration =((double)(stop-start))/CLK_TCK;//calculate duration for a single run of the
function
        printf("%16lf",duration);//output duration
        end=clock();
        que[i]=((double)(end-begin))/CLK_TCK;//calculate Total Time for a repetition
    }
    printf("\nTotal Time:                             ");
    for(int i=0;i<8;i++){
        printf("%16lf",que[i]);//output each Total Time
    }
    return 0;
}
}
```

DECLARATION

I hereby declare that all the work done in this project titled "Project 1: **Performance Measurement (POW)** " is of my independent effort.