Lab1
Performance Measurement (POW)
Exploring Power Calculation Algorithms

Chapter 1: Introduction

Problem description: The problem involves finding two different algorithms for computing X^N, where X is a positive integer (base) and N is also a positive integer (exponent). The goal is to analyze the complexities of these two algorithms.

Algorithm 1:

The first algorithm computes X^N using a straightforward approach, which involves N-1 multiplications. This means that you repeatedly multiply X by itself N-1 times to obtain X^N.

Algorithm 2:

The second algorithm employs a recursive approach to compute X^N. It follows the following rules:

1)If N is even, it computes X^(N/2) and then multiplies it by itself, resulting in X^N = X^(N/2) * X^(N/2).

2)If N is odd, it first computes X^((N-1)/2) recursively and then multiplies it by itself, followed by an additional multiplication with X. This results in X^N = X^((N-1)/2) * X^((N-1)/2) * X.

The problem's main focus is to analyze and compare the time complexities of these two algorithms. Time complexity is a measure of how the runtime of an algorithm scales with the size of the input, N in this case.Then we analyze the results and determine whether they match the time complexity of the respective algorithms we analyzed.

Background of the algorithms: To some extent,the 2nd algorithm is just like the binary search,which separates the initial problem in 2 pieces,solve each of them and eventually merge them to get the final results.

Chapter 2: Algorithm Specification

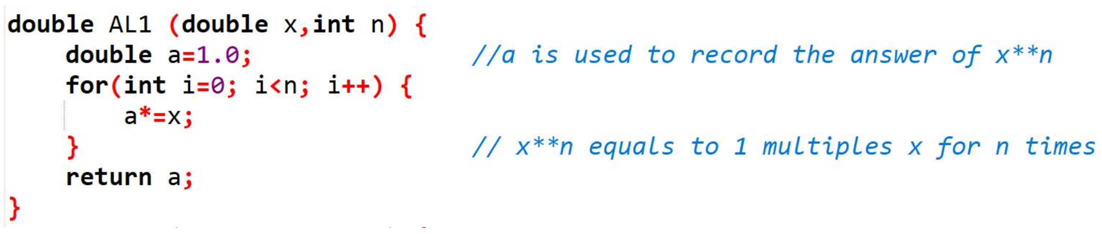
Algorithm 1

The first algorithm computes X^N using a straightforward approach, which involves N-1 multiplications. This means that you repeatedly multiply X by itself N-1 times to obtain X^N.

Pseudo Code

```
procedure Algorithm1 (x:double;n:integer)
    power:=1
        for i:=1 to n do
            power:=power*x
        end
    return power
```

Original Code



```
double AL1 (double x,int n) {
    double a=1.0;              //a is used to record the answer of x**n
    for(int i=0; i<n; i++) {
        a*=x;
    }                          // x**n equals to 1 multiples x for n times
    return a;
}
```

Algorithm 2(iterative)

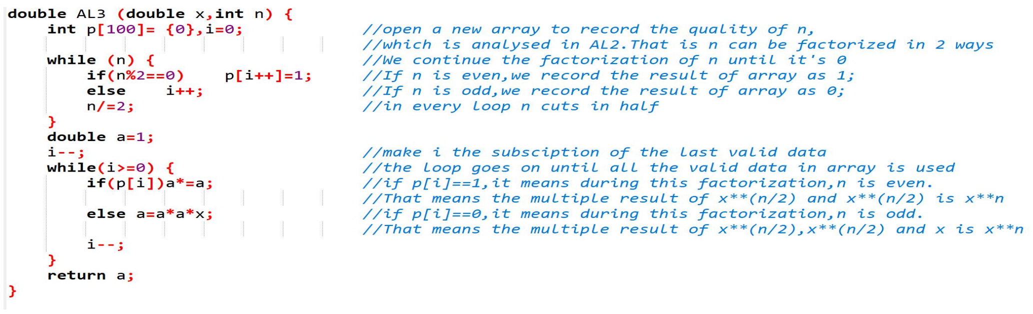
The second algorithm uses a iterative approach to compute X^N. If N is even, it records ‘1’ in an array. If N is odd, it records ‘0’ instead.Then n cuts in half.We loop this process until n is 0.

Then we start to compute the result according to the data stored in the array.If it’s 1,then compute the power by multiplying itself.If it’s 0,then the power equals to multiplying itself and finally multiplying X.We loop this process until every valid data in the array is used.

Pseudo Code

```
procedure Algorithm2(iteration) (x:double;n:integer)
    p[1...100]:={0}
    i:=0
    while (n>=0) do
        {
        if ( n%2=0) then
            {p[i]=1
            i:=i+1
            /}
        Else
            i:=i+1
        n:=n/2
        /}
    power:=1
    i:=i-1
    while (i>=0) do
        {
        If (p[i]=1)then
            power:=power*power
        else
            power:=power*power*x
        i:=i-1
        /}
Return power
```

Original Code



```
double AL3 (double x,int n) {
    int p[100]= {0},i=0;                  //open a new array to record the quality of n,
                                          //which is analysed in AL2.That is n can be factorized in 2 ways
    while (n) {                           //We continue the factorization of n until it's 0
        if(n%2==0)    p[i++]=1;           //If n is even,we record the result of array as 1;
        else    i++;                      //If n is odd,we record the result of array as 0;
        n/=2;                             //in every loop n cuts in half
    }
    double a=1;
    i--;                                  //make i the subsciption of the last valid data
    while(i>=0) {                         //the loop goes on until all the valid data in array is used
        if(p[i])a*=a;                     //if p[i]==1,it means during this factorization,n is even.
                                          //That means the multiple result of x**(n/2) and x**(n/2) is x**n
        else a=a*a*x;                     //if p[i]==0,it means during this factorization,n is odd.
                                          //That means the multiple result of x**(n/2),x**(n/2) and x is x**n
        i--;
    }
    return a;
}
```

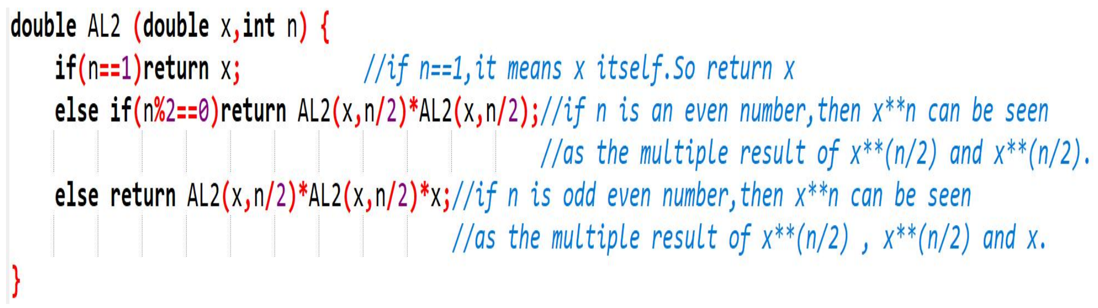
Algorithm 2(recursive)

The third algorithm uses a recursive approach to compute X^N. If N is even, it recursively computes X^(N/2) and then multiplies it with itself. If N is odd, it recursively computes X^(N/2), then multiplies it with itself, and finally multiplies the result by X.

Pseudo Code

```
procedure Algorithm2(recursive) (x:double;n:integer)
    if (n=0) then
        Return x
    Else if (n%2=0) then
        Return Algorithm2(recursive) (x,n/2) *Algorithm2(recursive) (x,n/2)
    Else
        Return Algorithm2(recursive) (x,n/2) *Algorithm2(recursive) (x,n/2)*x
```

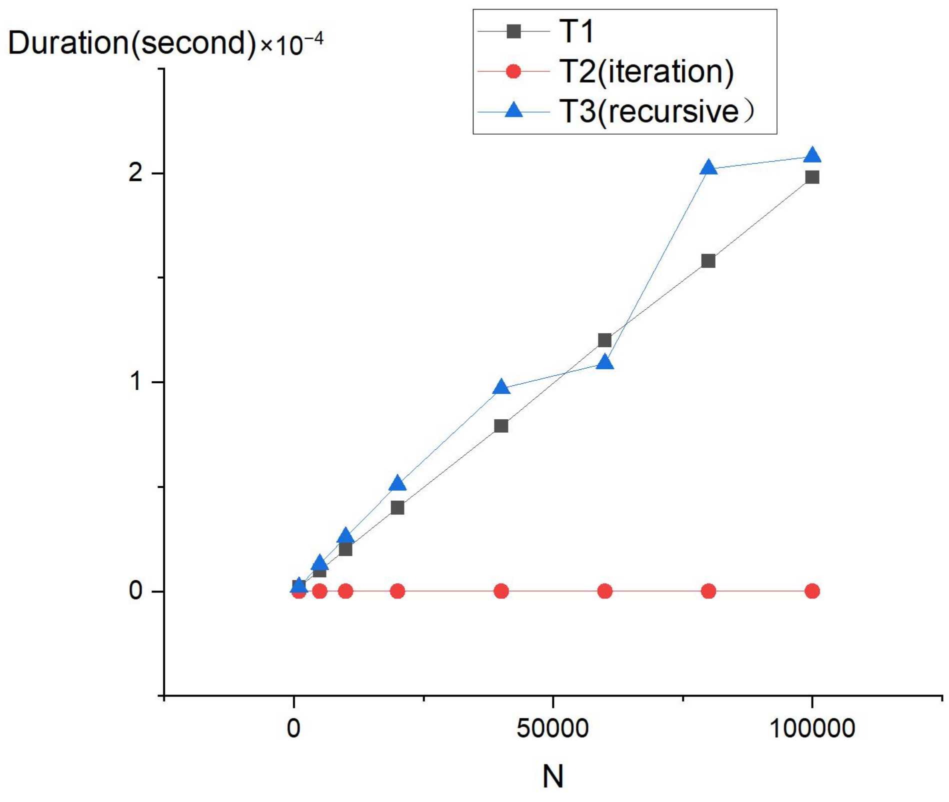
Original Code



```
double AL2 (double x,int n) {
    if(n==1)return x;          //if n==1,it means x itself.So return x
    else if(n%2==0)return AL2(x,n/2)*AL2(x,n/2);//if n is an even number,then x**n can be seen
                                                  //as the multiple result of x**(n/2) and x**(n/2).
    else return AL2(x,n/2)*AL2(x,n/2)*x;//if n is odd even number,then x**n can be seen
                                        //as the multiple result of x**(n/2) , x**(n/2) and x.
}
```

Chapter 3: Testing Results

	N	1E+3	5E+3	1E+4	2 E+4	4 E+4	6 E+4	8 E+4	1 E+5
Algorithm1	Iteration (K)	1E+5	5E+4	5E+4	2E+4	1E+4	5E+3	2E+3	2E+3
	Ticks	195	491	988	792	794	598	316	395
	Total Time(sec)	0.195	0.491	0.988	0.792	0.794	0.598	0.316	0.395
	Duration (sec)	2E-6	1E-5	2E-5	4E-5	7.9E-5	1.2E-4	1.58E-4	1.98E-4
Algorithm2 (iterative version)	Iteration (K)	5E+7	2E+7	1E+7	1E+7	1E+7	1E+7	1E+7	2E+7
	Ticks	2284	1049	557	589	612	637	650	1302
	Total Time(sec)	2.284	1.049	0.557	0.589	0.612	0.637	0.65	1.302
	Duration (sec)	4.57E-8	5.24E-8	5.57E-8	5.89E-8	6.12E-8	6.37E-8	6.50E-8	6.51E-8
Algorithm3 (recursive version)	Iteration (K)	1E+5	5E+4	5E+4	2E+4	1E+4	5E+3	2E+3	2E+3
	Ticks	160	638	1280	1023	974	547	403	416
	Total Time(sec)	0.16	0.638	1.28	1.023	0.974	0.547	0.403	0.416
	Duration (sec)	2E-6	1.3E-5	2.6E-5	5.1E-5	9.7E-5	1.09E-4	2.02E-4	2.08E-4



Chapter 4:　Analysis and Comments

Analysis of Complexity:

(1) Algorithm1:

Each multiplication operation takes constant time. Therefore, the time complexity of Algorithm 1 is **O(N)**.

(2) Algorithm2(iterative version):

In this algorithm,computer first performs log2(N) simple divisions to fill the array , and then does at most 2*log2(N) simple multiplications to get the final result.(When the data is 1,we do only 1 multiplication;when the data is 0,we do 2 multiplications.)Thus the computer calculates at most 3*log2(N) times before gaining the result,which means the time complexity of this algorithm is **log(N)**.

(3) Algorithm2(recursive version):

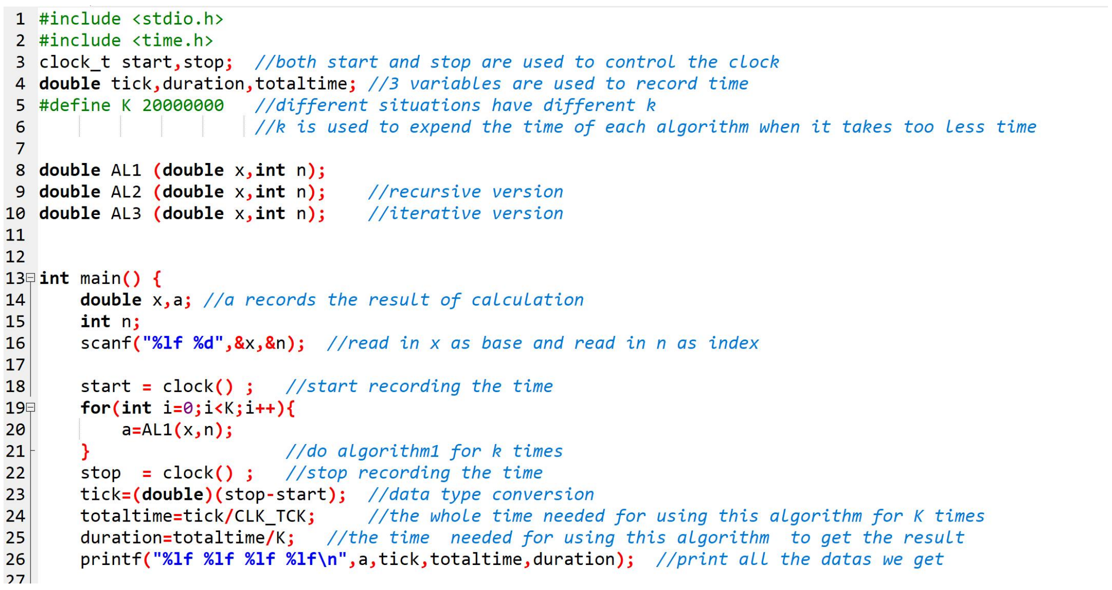
In the case when N is even, the algorithm performs 2 recursive call on N/2 and then performs a single multiplication.Therefore, for even N, T(N) can be expressed as:T(N) = 2*T(N/2) + 1.

In the case when N is add, the algorithm performs 2 recursive call on N/2 and then performs 2 single multiplications.Therefore, for even N, T(N) can be expressed as:T(N) = 2*T(N/2) + 2.

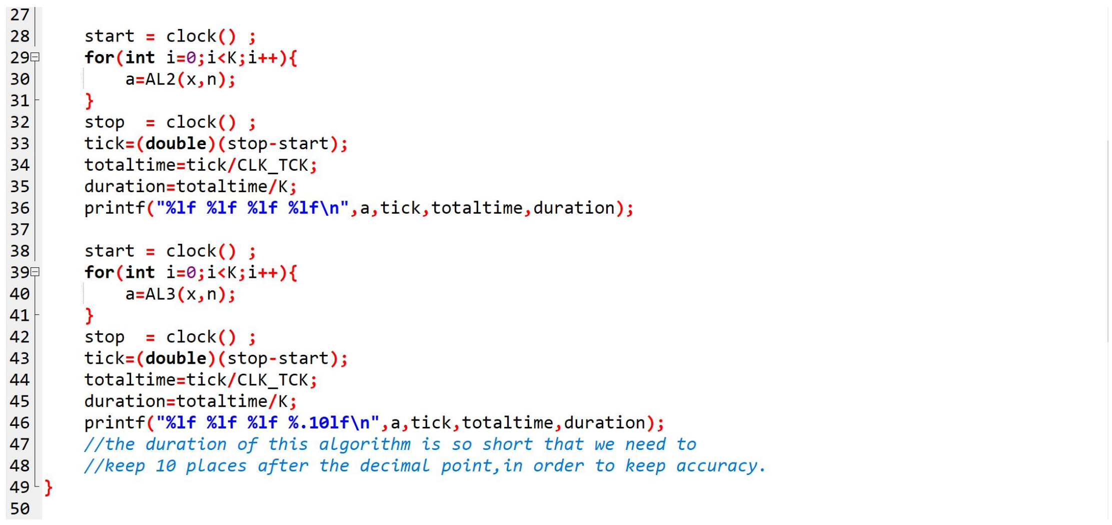
Thus the time complexity of this algorithm is **O(N)**.

Comments: Since the time complexity of the second algorithm has reached **log(N)**, it is theoretically difficult to find a more optimized algorithm.

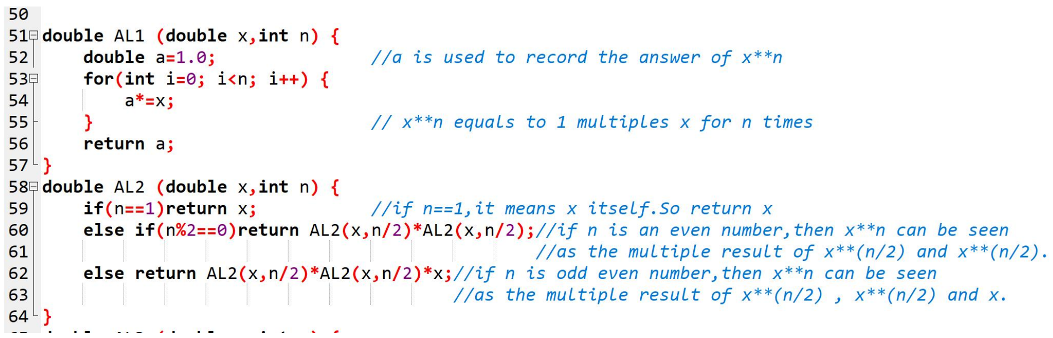
Appendix: Source Code (in C)



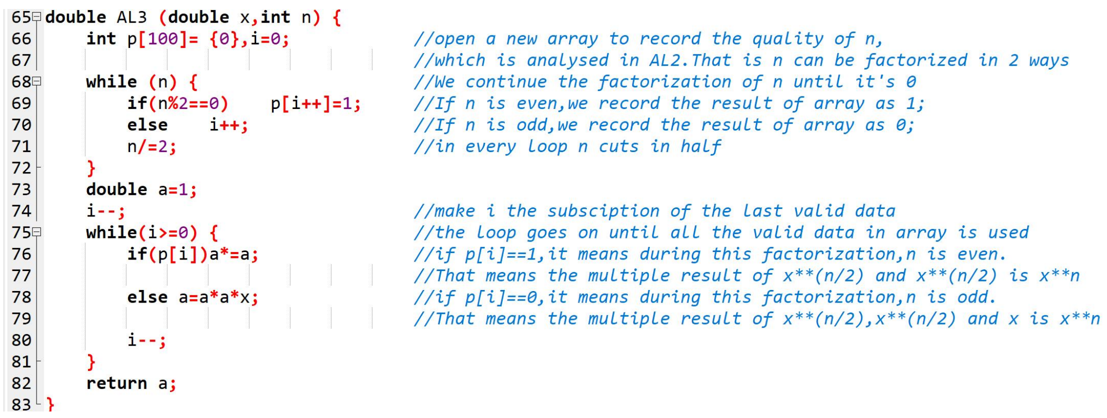
```c
#include <stdio.h>
#include <time.h>
clock_t start,stop;  //both start and stop are used to control the clock
double tick,duration,totaltime; //3 variables are used to record time
#define K 20000000   //different situations have different k
                     //k is used to expend the time of each algorithm when it takes too less time

double AL1 (double x,int n);
double AL2 (double x,int n);    //recursive version
double AL3 (double x,int n);    //iterative version


int main() {
    double x,a; //a records the result of calculation
    int n;
    scanf("%lf %d",&x,&n);  //read in x as base and read in n as index

    start = clock() ;   //start recording the time
    for(int i=0;i<K;i++){
        a=AL1(x,n);
    }                   //do algorithm1 for k times
    stop  = clock() ;   //stop recording the time
    tick=(double)(stop-start);  //data type conversion
    totaltime=tick/CLK_TCK;     //the whole time needed for using this algorithm for K times
    duration=totaltime/K;   //the time  needed for using this algorithm  to get the result
    printf("%lf %lf %lf %lf\n",a,tick,totaltime,duration);  //print all the datas we get
```



```c

    start = clock() ;
    for(int i=0;i<K;i++){
        a=AL2(x,n);
    }
    stop  = clock() ;
    tick=(double)(stop-start);
    totaltime=tick/CLK_TCK;
    duration=totaltime/K;
    printf("%lf %lf %lf %lf\n",a,tick,totaltime,duration);

    start = clock() ;
    for(int i=0;i<K;i++){
        a=AL3(x,n);
    }
    stop  = clock() ;
    tick=(double)(stop-start);
    totaltime=tick/CLK_TCK;
    duration=totaltime/K;
    printf("%lf %lf %lf %.10lf\n",a,tick,totaltime,duration);
    //the duration of this algorithm is so short that we need to
    //keep 10 places after the decimal point,in order to keep accuracy.
}

```



```c

double AL1 (double x,int n) {
    double a=1.0;               //a is used to record the answer of x**n
    for(int i=0; i<n; i++) {
        a*=x;
    }                           // x**n equals to 1 multiples x for n times
    return a;
}
double AL2 (double x,int n) {
    if(n==1)return x;           //if n==1,it means x itself.So return x
    else if(n%2==0)return AL2(x,n/2)*AL2(x,n/2);//if n is an even number,then x**n can be seen
                                                //as the multiple result of x**(n/2) and x**(n/2).
    else return AL2(x,n/2)*AL2(x,n/2)*x;//if n is odd even number,then x**n can be seen
                                        //as the multiple result of x**(n/2) , x**(n/2) and x.
}
```



```c
double AL3 (double x,int n) {
    int p[100]= {0},i=0;            //open a new array to record the quality of n,
                                    //which is analysed in AL2.That is n can be factorized in 2 ways
    while (n) {                     //We continue the factorization of n until it's 0
        if(n%2==0)    p[i++]=1;     //If n is even,we record the result of array as 1;
        else    i++;                //If n is odd,we record the result of array as 0;
        n/=2;                       //in every loop n cuts in half
    }
    double a=1;
    i--;                            //make i the subsciption of the last valid data
    while(i>=0) {                   //the loop goes on until all the valid data in array is used
        if(p[i])a*=a;               //if p[i]==1,it means during this factorization,n is even.
                                    //That means the multiple result of x**(n/2) and x**(n/2) is x**n
        else a=a*a*x;               //if p[i]==0,it means during this factorization,n is odd.
                                    //That means the multiple result of x**(n/2),x**(n/2) and x is x**n
        i--;
    }
    return a;
}
```

Declaration

I hereby declare that all the work done in this project is of my independent effort.