Fundamental Data Structure Project 1 Report

Performance Measurement(POW)

Team 26

Date:2019-10-8

Contents

Part1: Introduction to the project

There are at least two algorithms that can compute X^N for positive integer N. One is to use N-1 multiplications and another is to compute $X^N=X^{N/2}*X^{N/2}$ for even N and $X^N=X^{(N-1)/2}*X^{(N+1)/2}$ for odd N, which could be implemented in two methods, Iteration and Recursion.

In this project, we will implement the 3 functions of two algorithms and a testing program to test the time the functions need to execute the algorithms. We will analysis the time and space complexity of those algorithms and the result we get in the experiment to make some conclusions about computing X^N.

Part2: Algorithm and Program Instructions

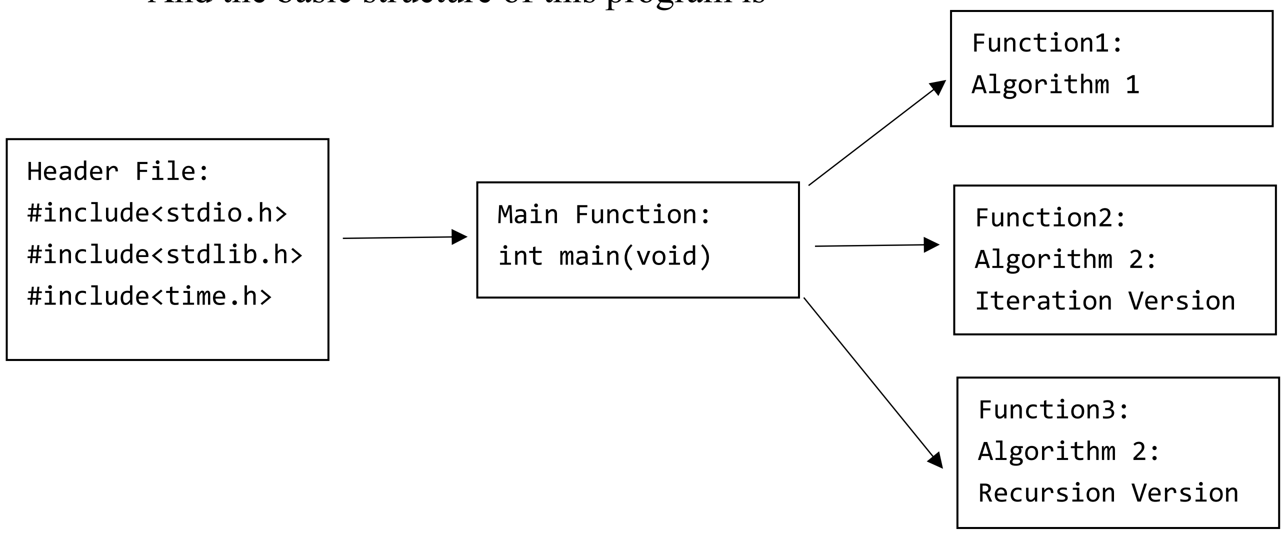
Now we will show the pseudocode of the functions:

```
ALGORITHM_1
procedure algorithm1(X: the power base of the calculation, N: the
exponent)
result :=x
for i := 1 to N-1
    result *= X
return result
```

```
ALGORITHM_2_1_Iteration
procedure algorithm2_1(X: the power base of the calculation, N: the
exponent)
result :=1
x = X
while N>0:
    if(N%2==1) result = result * X
    N = N/2
    x = x * x
return result
```

```
ALGORITHM_2_2_Recursion
procedure algorithm2_2(X: the power base of the calculation, N: the
exponent)
result :=1
if N==1
    return X
else if N%2==1
    return algorithm2_2(X, (N-1)/2) *algorithm2_2(X, (N+1)/2)
else
```

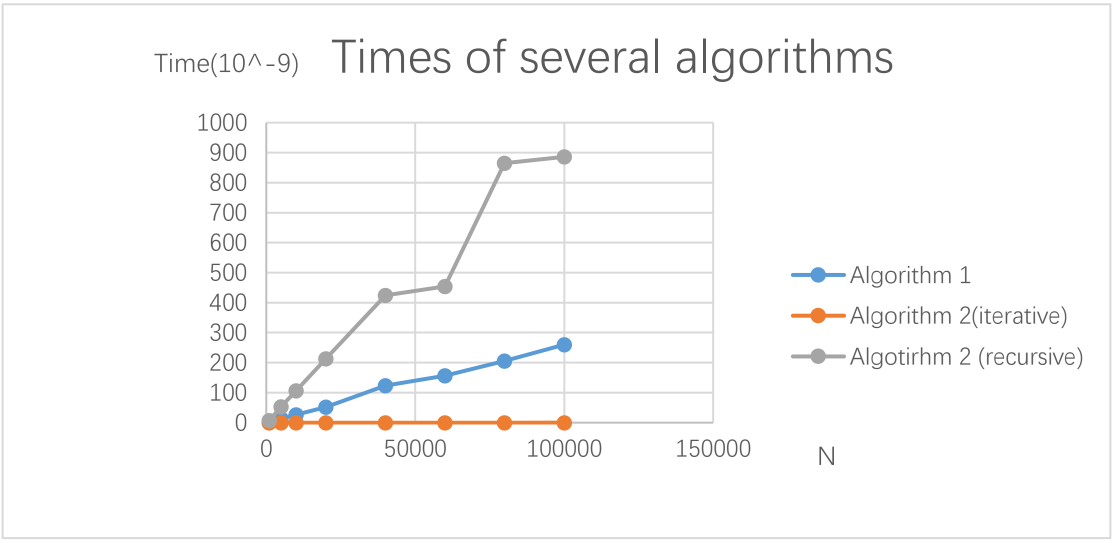
And the basic structure of this program is



Part3: Testing Results with Analysis and Comments

N	1000	5000	10000	20000	40000	60000	80000	100000
Iterations(K)	1000000	10000	10000	5000	2000	1000	1000	500
Ticks	2596	130	260	258	207	156	206	130
Total Time(sec)	2.60E-03	1.30E-04	2.60E-04	2.58E-04	2.47E-04	1.56E-04	2.06E-04	1.30E-04
Duration(sec)	2.60E-09	1.30E-08	2.60E-08	5.16E-08	1.24E-07	1.56E-07	2.06E-07	2.60E-07
Iterations(K)	1000000	500000	1000000	1000000	1000000	1000000	10000000	15000000
Ticks	35	21	45	48	50	51	528	794
Total Time(sec)	3.50E-05	2.10E-05	4.50E-05	4.80E-05	5.00E-05	5.10E-05	5.28E-04	7.94E-04
Duration(sec)	3.5E-11	4.2E-11	4.5E-11	4.8E-11	5E-11	5.1E-11	5.28E-11	5.29E-11
Iterations(K)	1000000	10000	5000	5000	2000	1000	1000	500
Ticks	7204	532	530	1064	849	454	865	443
Total Time(sec)	7.20E-03	5.32E-04	5.30E-04	1.06E-03	8.49E-04	4.54E-04	8.65E-04	4.43E-04
Duration(sec)	7.204E-09	5.32E-08	1.06E-07	2.128E-07	4.25E-07	4.54E-07	8.65E-07	8.86E-07

This is the testing results of 3 methods to compute X^N and we have drew a graph about the using time of three methods. From the table and the graph, we can easily find that the time-costing of the iterative version of algorithm 2 grows slowly as N become larger, which is in line with the growth trend of O(logN). And we can also find that the time-costing of algorithm 1 grows more quickly than iterative version of algorithm 2 , which is in line with the growth trend of O(N). Finally, the time-costing of the recursive version of algorithm 2 is the largest, maybe because its space complexity is not O(1). We will make further effort to analysis the time and space complexity of three algorithms.



Part4: Analysis of Algorithm and Codes

I. Why we choose an iteration of k?

For computers, execute just once to computer X^N takes too little time that we could hardly measure the time. So we set K like 1000 or 10000 as the number of the same iteration that the program could execute more times. When we analysis the data we get in the experiment, we will calculate the average of one time the algorithm need to make some conclusion.

All in all, it is important to choose a good number of K.

II. Time and space complexity of the algorithms

In algorithm 1, we have N-1 times of multiplies and all the main operations are on the variable "result", so the time complexity of algorithm 1 is O(N) and the space complexity is O (1).

In method 1 of algorithm 2, we use iteration in it and have that $X^{2N+1}=X^{2N}*X$ and $X^{2N}=X^{N}*X^{N}$, and no other n-dimension variable is used, so the time complexity is O(log N) and space complexity is O(1).

In method 2 of algorithm 2,we use recursion in it and have function relations that f(n)=f(n/2)*f(n/2)for even n and f(n)=f((n+1)/2)* f((n-1)/2) for odd n, besides the recursion call for the same function for many times which need O(log N) memory space to execute. So the time complexity is still O(log n) but the space complexity becomes O(log n).

We have made a simple table for the COMPLEXITY of 2 algorithms

ALGOTITHM	TIME COMPLEXITY	SPACE COMPLEXITY
1	O(N)	O(1)
2_1 ITERATION	O(Log N)	O(1)
2_2 RECURSION	O(Log N)	O(Log N)

III. Comparison between algorithm 1 and 2

We know that the time the functions need to execute the result depends on the time complexity of the algorithm. Algorithm 1 is O(N) and two methods of Algorithm 2 are O(log N) , so the time_2 is always much less than time_1. However, according to the results of our project, when it comes to method Recursion of Algorithm 2, time_2 of it is always much longer than time_1. So, what contributes to this result?

IV. Comparison between Iteration and Recursion

Algorithm 2 has two methods: Iteration and Recursion. Though the time complexity of the two are similar, but the space complexity of Recursion is much larger than Iteration, which we think make Recursion takes more time than Iteration. Because Recursion takes too much time to call the functions again and again and allocate memory also need plenty of time.

Part5: Source code

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

typedef double ElementType;

//record the start time and the end time
clock_t start1, stop1, start2, stop2, start3, stop3;

//record the run time of three functions
double duration1, duration2, duration3;
```

```c
/*Algorithm1, using N-1 multiplications to compute the res
ult directly*/
ElementType Result1(ElementType X, int N);

/*Iteration Version of Algorithm 2*/
ElementType Result2(ElementType X, int N);

/*Recursion Version of Algorithm 2*/
ElementType Result3(ElementType X, int N);

int main()
{
    // X is the base number
    ElementType X;

    // N is exponent of X
    // K is how many times the function repeat
    int N, K;
    scanf("%lf%d%d", &X, &N, &K);

    // record the result of different algorithms
    double result1, result2, result3;
    double tick1, tick2, tick3;

    // record how long the Algorithm 1 takes
    start1 = clock();
    for(int i = 1; i <= K; i++) {
        result1 = Result1(X, N);
    }
    stop1 = clock();

    // record how long the iteration version of Algorithm
2 takes
    start2 = clock();
    for(int i = 1; i <= K; i++) {
        result2 = Result2(X, N);
    }
    stop2 = clock();
    // record how long the recursion version of Algorithm
2 takes
    start3 = clock();
    for(int i = 1; i <= K; i++) {
        result3 = Result3(X, N);
    }
    stop3 = clock();

    // print the results respectively
    printf("Result1:%f\tResult2:%f\tResult3:%f\n", result1
, result2, result3);

    // get the number of the total ticks that takes to fin
ish the algorithm respectively
    tick1 = stop1 - start1;
    tick2 = stop2 - start2;
    tick3 = stop3 - start3;

    // divide ticks by CLK_TCK to get the total time respe
ctively
    duration1 = ((double)(tick1))/CLK_TCK;
    duration2 = ((double)(tick2))/CLK_TCK;
    duration3 = ((double)(tick3))/CLK_TCK;

    // print the total ticks respectively
    printf("Tick1: %f\tTick2: %f\tTick3: %f\n", tick1, tic
k2, tick3);

    // print the total time respectively
    printf("Time1: %fms\tTime2: %fms\tTime3: %fms\n", dura
tion1, duration2, duration3);

    system("pause");
    return 0;
}

/**************************************************
 * Function:  Result1
 * Description:  Algorithm1, using N-1 multiplications to
compute the result directly
```

```
 * Input:     X is the base number, N is the exponent of X

 * Return:    the result of N the power of X
 * ************************************************/
ElementType Result1(ElementType X, int N)
{
    double result = X;
    for(int i=1; i<=N-1; i++) {
        result *= X;
    }
    return result;
}

/************************************************
 * Function:  Result2
 * Description:  Iteration Version of Algorithm 2.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.
 * Input:     X is the base number, N is the exponent of X

 * Return:    the result of N the power of X
 * ************************************************/
ElementType Result2(ElementType X, int N)
{
    double result = 1;
    double x=X;
    while(N>0) {
        if(N%2 == 1) result = result*x;
        N = N/2;
        x = x*x;
    }
    return result;
}

/************************************************
 * Function:  Result3
 * Description: Recursion Version of Algorithm 2.If N is e
ven, we will return Result3(X,N/2)*Result3(X,N/2); and if
N is odd, we will return Result3(X, (N-1)/2) *Result3(X, 
(N-1)/2) *X.
```

```
 * Input:     X is the base number, N is the exponent of X
 * Return:    the result of N the power of X
 * *************************************************/
ElementType Result3(ElementType X, int N)
{
    if(N == 1) return X;
    if(N%2 == 0) return Result3(X,N/2)*Result3(X,N/2);
    else return Result3(X,(N-1)/2)*Result3(X,(N-1)/2)*X;
}
```

Part6: Summary

In conclusion, Project 1 is to deal with the question on comparing the complexities of the 2 algorithms which are expressed via 1 and 2 functions, respectively. The method of iteration and recursion is always our powerful weapon to divide complex questions into easier ones.

Project 1 provides us a chance to penetrate the analysis of algorithm in both time complexity and space complexity.