Fundamentals of Data Structures

Laboratory Project

1 Performance Measurement (POW)

Group 18

Authors:严子涵 张宇晴 张雯琪

Date：2019-09-27

Chapter 1: Introduction

1.1 Problem description

This project concerns with 3 parts.

Firstly we shall carry out 2 algorithms to compute X^N for some positive integer N.

Secondly we shall analyze time and space complexities between the two algorithms.

Thirdly we shall finish the iterative and recursive implementations of algorithm 2, and give the measurement and comparison of their performances with the help of given C's standard library time.h.

1.2 Background of the algorithms.

Algorithm1 is to use N-1 multiplications

Algorithm2:

If N is even, X^N = X^N/2 × X^N/2

If N is odd, X^(N-1)/2 x X^(N-1) x X

For the testing program we may use C's standard library time.h

Chapter 2: Algorithm Specification

2.1Description of three algorithms, in the form of pseudo-code:

2.1.1Algorithm 1:

```
function AL1(x, positiveintegern)
    for i = 0 → n − 1 do
        result ← result ∗ x
    end for
    return result
end function
```

First define the result value/return value and initialize it, then multiply the initial return value by x for n times to get the result.

2.1.2Algorithm 2 Iterative version:

```
function Al2_I(x, positiveintegern)
    result ← x
    while n > 1 do
        if n = 3 then
            result ← result ∗ result ∗ x
        else
            result ← result ∗ result
        end if
        if n|2 = 0 then
            n ← n/2
        else
            n ← n/2 + 1
        end if
    end while
    return result
end function
```

For two occasions that n is odd or even, we divide n to half part or plus one else. And multiply the temporal result each time to get the final answer.

2.1.3Algorithm 2 Recursive version:

8: **function** $Al2_R(x, positiveintegern)$
9: **if** $n = 1$ **then**
10: $result \leftarrow x$
11: **else**
12: **if** $n|2 = 0$ **then**
13: $tmp \leftarrow Al2_R(x, n/2)$
14: $result \leftarrow tmp * tmp$
15: **else**
16: $tmp \leftarrow Al2_R(x, n/2)$
17: $result \leftarrow tmp * tmp * x$
18: **end if**
19: **end if**
20: **return** $result$
21: **end function**

For two occasions that n is odd or even, we multiply the temporal result by itself or plus x else, and the recursive depth each time is n/2.

2.2Description of the test algorithm:

Firstly we record the time before the function runs

Then we run the function and call it by k times

At the end of the function loop, we record the end time

The difference between the end time and the start time is the Ticks

According to the duration = ((double) (stop - start))/CLK_TCK; The runtime = duration/k;

We can find the value of duration and runtime

Then we adjust the value of k according to the three values of the output through comparative analysis of different k algorithm analysis

Chapter 3: Testing Results

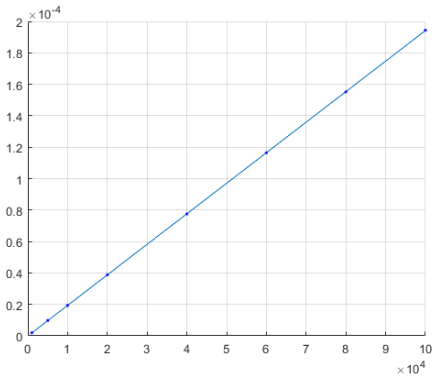
The testing result of measuring and comparing the performances of Algorithm 1 and the iterative and recursive implementations of Algorithm 2 for X=1.0001 and N = 1000, 5000, 10000, 20000, 40000, 60000, 80000, 100000.

	N	1000	5000	10000	20000	40000	60000	80000	100000
Algorithm 1	Iterations	1.00E+05	1.00E+05	1.00E+05	1.00E+05	1.00E+05	1.00E+05	1.00E+05	1.00E+05
	Ticks	196	975	1939	3877	7754	11640	15519	19442
	Total Time (s)	0.196	0.975	1.939	3.877	7.754	11.64	15.519	19.442
	Duration (s)	1.960E-06	9.750E-06	1.939E-05	3.877E-05	7.754E-05	1.164E-04	1.552E-04	1.944E-04
Algorithm 2 (Iterative version)	Iterations	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08
	Ticks	1857	2144	2327	2535	2696	3000	3072	3167
	Total Time (s)	1.857	2.144	2.327	2.535	2.696	3	3.072	3.167
	Duration (s)	1.857 e-8	2.144 e-8	2.327 e-8	2.535 e-8	2.696 e-8	3.000 e-8	3.072 e-8	3.167 e-8
Algorithm 2 (Recursive version)	Iterations	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08	1.00E+08
	Ticks	4594	6224	6780	7252	7692	7902	8249	8309
	Total Time (s)	4.594	6.224	6.78	7.252	7.692	7.902	8.249	8.309
	Duration (s)	4.59E-08	6.224 e-8	6.780 e-8	7.252 e-8	7.692 e-8	7.902 e-8	8.249 e-8	8.309 e-8

Chapter 4: Analysis and Comments

4.1Algorithms analysis (fitting function images):

4.1.1Algorithm 1

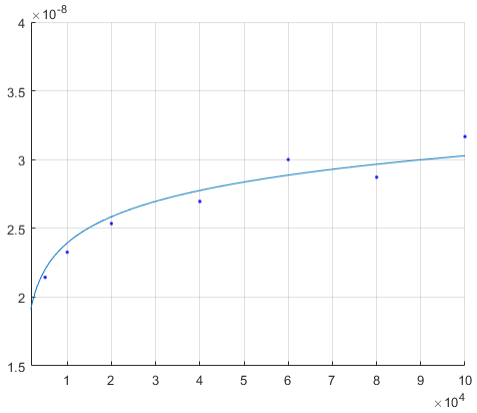


We use MATLAB to calculate the fit function and show the result below:

t1 = 1.9424e-09 * N - 4.8176e-08 with R-squared = 1.0000

This is a linear function which proves that algorithm 1 has time complexity O(N). The reason is that this algorithm does N loops and for every loop it will compute 3 times. So totally it will compute 3N+1 times.

4.1.2Algorithm 2(Iterative version)

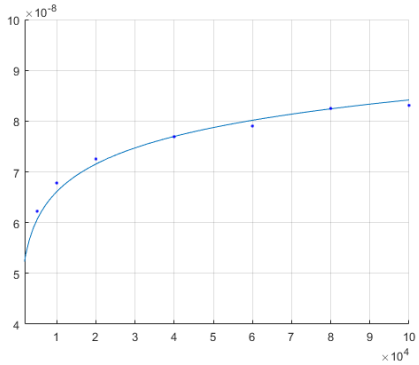


t2 = 2.7528e-09 * log(N) - 1.4160e-09 with R-squared =0.9940

It's obvious to learn that this algorithm has time complexity of O(log N).

The reason is that for the worst case, n is divided into halves until it reaches 1. If N<=2^I, this algorithm will do at most I loops and for every loop it will compute constant times, resulting a log N time limit.

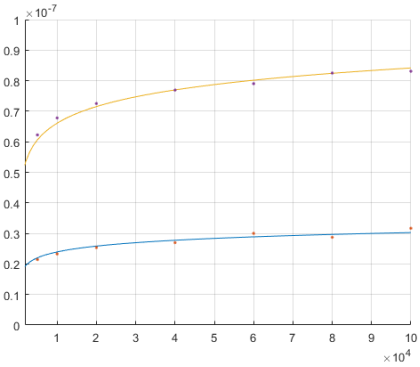
4.1.3Algorithm 2(Recursive version)



t3 = 7.8453e-09 * log(N) - 6.1609e-09 with MSE R-squared = 0.9757

The recursive algorithm, though remarkably less efficient than its iterative version, also presents O(log N) complexity. The reasoning is pretty straight forward: the algorithm calls itself at most I loops as above and during each call it does constant complexity computation.

4.2Comparison between Iterative and Recursive version



The graph above compares time consumption of the iterative and recursive algorithms. Both functions have O(log n) time complexity while the recursive one requires much more (2 to 3 times) time than the iterative algorithm, which is largely because of huge costs of freq uent function calls in recursion.

4.3Conclusion:

4.3.1Time complexities of the three algorithms:

A1 O(n) 3n+1 (one comparison and two operations per cycle + the last comparison of i and n)

A2_Iteration log(n) in the worst case, n is divided into half part until it became 1, thus it complexity is log(n)

A3_Recursion log(n) in the worst case, n traces back to its half part each time after multiplication, so the same as log(n) complexity

4.3.2Space complexities of the three algorithms:

A1 O(1) just three variables: result, x, n

A2_Iteration O(1) just three variables: result, x, n

A2_Recursion O(logn) Recursive depth, space required for each recursion is n/2, so the space complexity is o(logn)

Appendix: Source Code (in C)

AL1.c

```c
/*Function of algorithm1*/
double AL1(double x, int n){
    double result = 1;           //Define the result value/return value and initialize it
    int i;
    for(i = 0; i < n; i++){      //Multiply the initial return value by x for n times
        result = result * x;
    }
    return result;               //Return x to the n
}
```

AL2Iteration.c

```c
/*Algorithm2 iterative version*/
double AL2Iteration(double x, int n)
{
    double result = x;         //Set the return value and assign the value of x
    while(n > 1){
        if(n == 3) {
            //if N is odd, X(n) = X((n-1)/2) * X((n-1)/2) * X. So n equals 3 is a separate case
            result = result * result * x;
            break;
        }

        else result = result * result;
            //if N is even, X(n) = X(n/2) * X(n/2)
        if(n % 2 == 0) n = n / 2;
            //If n is even, n goes into 2 every time
        else n = n / 2 + 1;
            //If n is odd
    }
    return result;
}
```

AL2Recursion.c

```c
/*Algorithm2 recursive version*/
double AL2Recursion(double x, int n)
{
    double result;
    if(n == 1) result = x;
        /*if N is even, X(n) = X^(n/2) × X^(n/2)*/
    else if(n % 2 == 0)
    {
        double tmp = AL2Recursion(x, n / 2);
        result = tmp * tmp;
    }
        /* if N is odd, X(n) = X^((n-1)/2) × X^((n-1)/2) × X*/
    else
    {
        double tmp = AL2Recursion(x, n / 2);
        result = tmp * tmp * x;
    }
    return result;
}
```

Test.c

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



extern double AL1(double x, int n);           /*Function of algorithm1*/
extern double AL2Iteration(double x, int n);  /*Algorithm2 iterative version*/
extern double AL2Recursion(double x, int n);  /*Algorithm2 recursive version*/
clock_t start, stop;                          /*clock_t is a built-in type for processor time(ticks)*/
double duration;                             /*records the run time (seconds) of a function*/

int main(){
    double x;
    double result;                           /*Use result to record x to the n and initialize it*/
    double runtime, duration, total_time;
    int n;
    int k;
    int c;
    int Ticks;

    //The following three sections only call different functions with the same logic,
    //so our group takes the first section as an example to annotate the idea in detail
    //algorithm 1
    printf("please enter the base x and the exponent n for algorithm1\n");
    scanf("%lf%d", &x, &n);                    /*Read in the unknowns and indices*/
    printf("please enter the iterative time k:\n");
    scanf("%d", k);                           /*we may repeat the function calls for K times to obtain a total run time*/
    start = clock();                          /*records the ticks at the beginning of the function call*/
    for(c = 1; c <= k; c++){                       /*repeat the function calls for K times*/
        result = AL1(x, n);
    }
    stop = clock();                            /*records the ticks at the end of the function call*/
    Ticks = stop - start;                     /*Calculate the ticks*/
    total_time = ((double)(stop - start)) / CLK_TCK; /*Calculate the total_time*/
    duration = total_time / k;                     /*Calculate the duration*/
    printf("result = %lf\n", result);          /*Output the result*/
    printf("Ticks = %d\n", Ticks);     /*Output the Ticks*/
    printf("total_time = %lf\n", total_time);       /*Output the total_time*/
    printf("duration = %lf\n", duration);           /*Output the duration*/
```

```c
    //algorithm 2 iterative version
    printf("please enter the base x and the exponent n for algorithm2 iterative version\n");
    scanf("%lf%d", &x, &n);
    printf("please enter the iterative time k:\n");
    scanf("%d", &k);
    start = clock();
    for(c = 1; c <= k; c++){
        result = AL2Iteration(x, n);
    }
    stop = clock();
    Ticks = stop - start;
    total_time = ((double)(stop - start)) / CLK_TCK; /*Calculate the total_time*/
    duration = total_time / k;                     /*Calculate the duration*/
    printf("result = %lf\n", result);          /*Output the result*/
    printf("Ticks = %d\n", Ticks);     /*Output the Ticks*/
    printf("total_time = %lf\n", total_time);       /*Output the total_time*/
    printf("duration = %lf\n", duration);           /*Output the duration*/

    //algorithm 2 recursive version
    printf("please enter the base x and the exponent n for algorithm2 recursive version\n");
    scanf("%lf%d", &x, &n);
    printf("please enter the iterative time k:\n");
    scanf("%d", &k);
    start = clock();
    for(c = 1; c <= k; c++){
        result = AL2Recursion(x, n);
    }
    stop = clock();
    Ticks = stop - start;
        total_time = ((double)(stop - start)) / CLK_TCK; /*Calculate the total_time*/
    duration = total_time / k;                     /*Calculate the duration*/
    printf("result = %lf\n", result);          /*Output the result*/
    printf("Ticks = %d\n", Ticks);     /*Output the Ticks*/
    printf("total_time = %lf\n", total_time);       /*Output the total_time*/
    printf("duration = %lf\n", duration);           /*Output the duration*/
    getchar();
    return 0;
}
```

Group Work：

Programmer:严子涵

Implement the three functions and a testing program with sufficient comments.

Tester: 张宇晴

Decide the iteration number K for each test case and fill in the table of results. Plot the run times of the functions. Write analysis and comments.

Report Writer: 张雯琪

Write Chapter 1, Chapter 2, and finally a complete report. Also take charge of peer reviews and the final version of this report.

Declaration：

We hereby declare that all the work done in this project titled "zjk_2019G18_P1" is of our independent effort as a group.