Three ways to realize power functions

Author’s name

Date:2021/10/6

Chapter 1: Introduction

Power function(x^n) is a foundational function in programing. This project requires to use three different ways to realize the power function when n is limited to a positive integer and design a testing program to evaluate the running speed of these algorithms in different circumstances. After testing, an analysis of these algorithms and comments on how to improve them are made.

Chapter 2: Algorithm Specification

Algorithm 1 calculate the result by considering x^n as x multiplies itself by (n-1) times.
The following is the pseudo-code of algorithm 1.

```
ans ← x
for i ← 1 to (n-1)
do ans ← ans*x
return ans
```

Algorithm 2 works as in the following way: if N is even, x^n=x^(n/2)*x^(n/2), else, x^n=(x^n/2)*(x^(n/2))*x.
In this project, algorithm 2 have two version, the iterative version and the recursive version. The iterative version works as follow:
The iterative version calls the function only once. When n is larger than 1, it starts the loop that changes x to x^x and n to n/2. If n is odd, the result is multiplied by x. When n is equal to 1,the loop ends.
The following is the pseudo-code of algorithm 2's iterative version.

```
ans ← x
if(n=1)
then [return ans]
while (n>0)
do{
      if (n % 2 != 0)
      then[ans ← ans * x]
      x ← x * x;
      n ← n / 2
}
return ans
```

The recursive version functions by changing the parameters of the function(x←x*x,n←n/2) and multiply by x if n is odd, and call itself again until n is

equal to 1.
The following is the pseudo code of the recursive version.

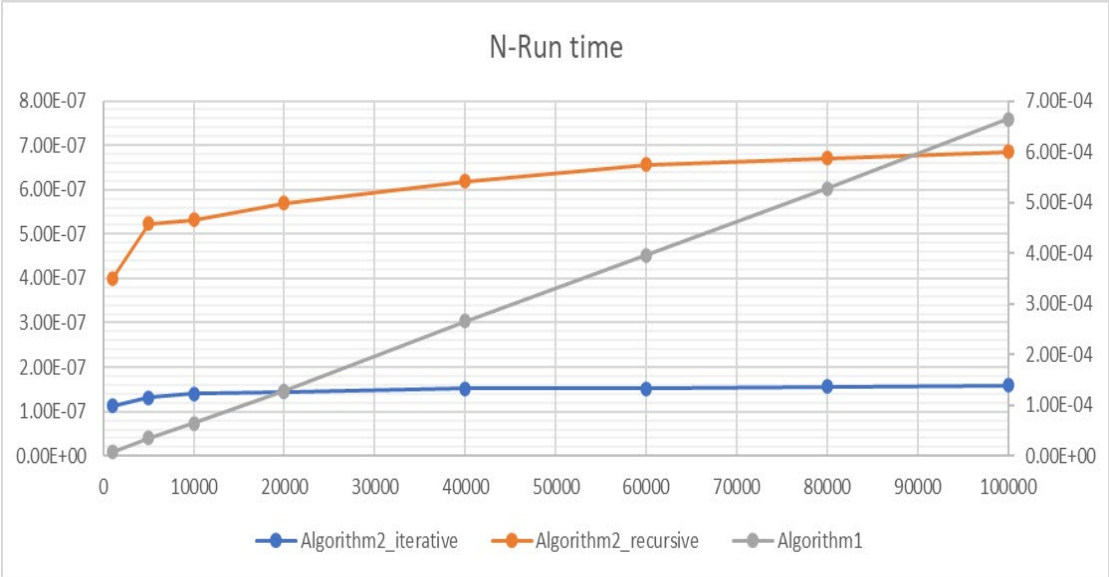
```
if (n=1)
then [return x]
if (n%2=0)
then [return algorithm2_recursive(x*x, n / 2)]
else [return algorithm2_recursive(x*x, n / 2)*x]
```

Chapter 3: Testing Results

To Compare the time complexity of the three algorithms, we use them to calculate 1.0001^N with different values of N, and record how many ticks and the total time are consumed when the program is running. As the running speed is too fast when the algorithm runs only once, we let it run for k times and divide the total time with k to get the duration of the algorithm. The following table records the test result.

	N	1000	5000	10000	20000	40000	60000	80000	100000
Algorithm 1	Iterations (K)	10^4	10^4	10^4	10^4	10^4	10^4	10^4	10^4
	Ticks	70	358	646	1283	2653	3963	5272	6635
	Total time (sec)	0.070	0.358	0.646	1.283	2.653	3.963	5.272	6.635
	Duration (sec)	7.0* 10^-6	3.58* 10^-5	6.46* 10^-5	1.283* 10^-4	2.653* 10^-4	3.963* 10^-4	5.272* 10^-4	6.635* 10^-4
Algorithm 2 (Iterative)	Iterations (K)	10^7	10^7	10^7	10^7	10^7	10^7	10^7	10^7
	Ticks	1130	1318	1398	1453	1524	1549	1558	1595
	Total time (sec)	1.130	1.318	1.398	1.453	1.524	1.517	1.558	1.595
	Duration (sec)	1.130* 10^-7	1.318* 10^-7	1.398* 10^-7	1.453* 10^-7	1.524* 10^-7	1.517* 10^-7	1.558* 10^-7	1.595* 10^-7
Algorithm 2 (recursive)	Iterations (K)	10^6	10^6	10^6	10^6	10^6	10^6	10^6	10^6
	Ticks	399	523	532	570	619	656	671	685
	Total time (sec)	0.399	0.523	0.532	0.570	0.619	0.656	0.671	0.685
	Duration (sec)	3.99* 10^-7	5.23* 10^-7	5.32* 10^-7	5.70* 10^-7	6.19* 10^-7	6.56* 10^-7	6.71* 10^-7	6.85* 10^-7

Then we use a N-Run-time graph to present the speed of the three algorithms.



The x-axis is the value of N, the y-axis on the left indicates run time of two versions of algorithm 2, the y-axis on the right indicates the run time of algorithm 1. Both have a unit of seconds.

Chapter 4: Analysis and Comments

The time complexity of algorithm 1 is O(N), as it uses (N-1) multiplications to calculate. The space complexity is O(1).
The time complexity of two version of algorithm is O(log n), as they divide n evenly. The space complexity of iterative version is O(1), and the recursive one is O(log n).
The time complexity is also shown in the graph, we can see that the graph of algorithm 1 is a linear one, and two versions of algorithm 2 are logarithmic curve.

Appendix: Source Code

The three algorithms are packed in the file “algorithm.c”.

```
#include"algorithm.h"

//Algorithm1 consider x^n as x multiply itself for n-1 times.
double algorithm1(double x, int n)
{
    double ans = x;                 //record the result of the multiplications
    int i;                          //record the number of the multiplications
    for (i = 1; i < n; i++)         //if the number of multiplications is less than n-
1,continue the loop.When the number is equal to n-1,end the loop.
    {
        ans = ans * x;
    }
    return ans;
}



//Algorithm 2 works in the following way:
//    if n is even and n>1,x^n=(x^(n/2))*(x^(n/2)).Else,x^n=(x^(n/2))*(x^(n/2))*x.
//The following two algorithms is the iterative and recursive version of algorithm 2.

//The iterative version
double algorithm2_iterative(double x, int n)
{
    double ans = x;                 //record the result of the calculation
    if (n == 1)                     //if n==1, then x^1==x.
    {
        return ans;
    }
    for (; n > 0;)                  //while n>0, continue the loop, else, end the loop.
    {
        if (n % 2 != 0)             //if n is odd, ans shall be multiplied by x.
        {
            ans = ans * x;
        }
        x = x * x;                  //change x to x*x, and change n to n/2 in the next
loop.
        n = n / 2;
    }
    return ans;
}
```

```
//The recursive version
double algorithm2_recursive(double x, int n)
{
    if (n == 1)                    //if n==1,the recursion ends.
    {
        return x;
    }
    if (n % 2 == 0)
    {
        return algorithm2_recursive(x*x, n / 2);        //if n is even,call the
function again with x replaced by x*x and n replaced by n/2 to do it recursively.
    }
    else
    {
        return algorithm2_recursive(x*x, n / 2)*x;      //if n is odd, a x should be
multiplied.
    }
}
```

The head file “algorithm.h” is coded to enable the use of the algorithms.

```
#pragma once
#ifndef __ALGORITHM_H__
#define __ALGORITHM_H__

double algorithm1(double x, int n);
double algorithm2_iterative(double x, int n);
double algorithm2_recursive(double x, int n);

#endif
```

The “main.c” can finish the test.

```
#define _CRT_SECURE_NO_WARNINGS                         //only used in visual studio to
avoid warnings of using "scanf" function.
#include<stdio.h>
#include<time.h>
#include"algorithm.h"

clock_t start, stop;                                    //record the starting ticks and
stopping ticks of the algorithm
double duration,                                        //record the running time of
the algorithm
       total_time,                                      //record the total running time
```

```
of the algorithm
       result;                                          //record the result of the
calculation
int k,                                                  //record the iteration times of
algorithm
    ticks;                                              //record total ticks when the
algorithm works

//print the result to the window
void output()
{
    ticks = (stop - start);                             //calculate the ticks by
start tick minus end tick
    total_time = ((double)(stop - start)) / CLK_TCK;    //divide ticks with CLK_TCK
to get total time
    duration = (((double)(stop - start)) / k) / CLK_TCK;   //divide total time with k
to get duration
    printf("Compute completed.\n\n");
    printf("The result is %lf\n", result);
    printf("The ticks are %d.\n", ticks);
    printf("The total time is %lf\n", total_time);
    printf("The duration is %lf\n", duration);
}

int main()
{
    double x;
    int n;
    int algo_selector;                                  //indicates the algorithm
selected
    int i;                                              //flag for the loop


    printf("Please input double x , positive interger n and iteration times (positive
intergers)k.\n");
    scanf("%lf %d %d", &x, &n, &k);                     //taking in the value of
x,n,k


    if (n <= 0||k<=0)                                   //check if the input
is valid
    {
        printf("Range Error,program ended.");
        return 1;
```

```
    }


    printf("Please enter a number to choose an algorithm\n");     //choose the
algorithm
    printf("1. algorithm1    2.algorithm2_iterative    3.algorithm2_recursive\n");
    scanf_s("%d", &algo_selector);

    if (algo_selector <= 0 || algo_selector > 3)                  //check if the
selection is valid
    {
        printf("Invalid selection, program ended!\n");
    }
    printf("Computing,please wait...\n");


    switch (algo_selector)                                        //use different
algorithms to calculate according to the selector
    {
    case 1:
        start = clock();                                          //start timing
        for (i = 0; i < k; i++)                                   //do for k times
        {
            result = algorithm1(x, n);
        }
        stop = clock();                                           //end timing
        output();                                                 //print the result
        break;

    case 2:
        start = clock();
        for (i = 0; i < k; i++)
        {
            result = algorithm2_iterative(x, n);
        }
        stop = clock();
        output();
        break;
    case 3:
        start = clock();
        for (i = 0; i < k; i++)
        {
            result = algorithm2_recursive(x, n);
```

```
        }
        stop = clock();
        output();
        break;
    default:
        break;
    }


}
```

Declaration

I hereby declare that
all the work done in this project titled "Three Ways to Realize Power Functions"
is of my independent effort.