Performance Measurement (POW)

黄文杰

Date：2023-9-27

Chapter 1: Introduction

There are two different algorithms that can compute X^N (N is a positive integer) . One algorithm is to use N−1 multiplications. Another algorithm 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 = $X^{(N-1)/2} \times X^{(N-1)/2} \times X$, where X^N means N power of X. We want to measure and compare the performances of the first algorithm and the iterative and recursive implementations of the second algorithm and analyze the complexities of the two algorithms.

Chapter 2: Algorithm Specification

- **pseudo-code of Algorithm1**

```
function PowOfAlgorithm1(x, n):
    result = 1
    for i from 1 to n:
        result = result * x
    return result
```

(Algorithm 1 works by using N-1 multiplications.)

- **pseudo-code of Algorithm2 (iterative version)**

```
function PowOfIterativeAlgorithm2(x, n):
    data[64]
    i = 0
    result = x
    if n == 0:
        return 1
    while n != 1:
        if n is odd:
            data[i] = 1
            i = i + 1
        else:
            data[i] = 0
            i = i + 1
        n = n >> 1
    for j from i-1 down to 0:
        if data[j] == 1:
            result = result * result * x
        else:
            result = result * result
    return result

```

(Algorithm 2(iterative version) first determines the binary representation of N, uses an array to record the information of each bit, and then determines the parity according to each binary number, so as to obtain the N power of X according to the formula.)

- **pseudo-code of Algorithm2 (recursive version)**

```
function PowOfRecursiveAlgorithm2(x, n):
    if n == 0:
        return 1
    else if n == 1:
        return x
    else if n is even:
        return PowOfRecursiveAlgorithm2(x * x, n / 2)
    else:
        return PowOfRecursiveAlgorithm2(x * x, (n - 1) / 2) * x

```

(Algorithm 2(recursive version) recursively obtains the Nth power of X (namely X^N) according to the formula : " if N is even, X^N = $X^{N/2} \times X^{N/2}$; and if N is odd, X^N = $X^{(N-1)/2} \times X^{(N-1)/2} \times X$ ".)

- **a sketch of the main program**

The main program is primarily divided into two parts: 1. Initializing global variables; 2. Entering a loop to await user input for specific functionalities (function categories can be seen in the screenshot below).

> - **First Part**
>
> ```
> // Initialize global variables
>     for(int i=0;i<2;i++){
>         k1[i]=10000;
>     }
>     for(int i=2;i<8;i++){
>         k1[i]=1000;
>     }
>     for(int i=0;i<8;i++){
>         k2[i]=1000000;
>         k3[i]=1000000;
>     }
> ```

> - **Second Part**
>
> ```
> // Loop waiting for user input
>     while(!is_exit){
>         int op; // Set test mode
>         ShowMenu1(); // Call a more aesthetically pleasing menu interface
> ```

```
        scanf("%d",&op);
        getchar();
        printf("\n");
        switch(op){
            case 1:
                // Manual testing
                ManualShowTest();
                break;
            case 2:
                // Automated testing
                AutoShowTest();
                break;
            case 3:
                // Modify configuration items for automation testing
                UpdateGlobalValueOfK();
                break;
            case 4:
                // Output averages based on automation testing results and counts
                ShowAverage();
                break;
            case 5:
                // Exit
                is_exit = true;
                break;
            default:
                printf("请输入正确的数字! \n");
                break;
        }
    }
```

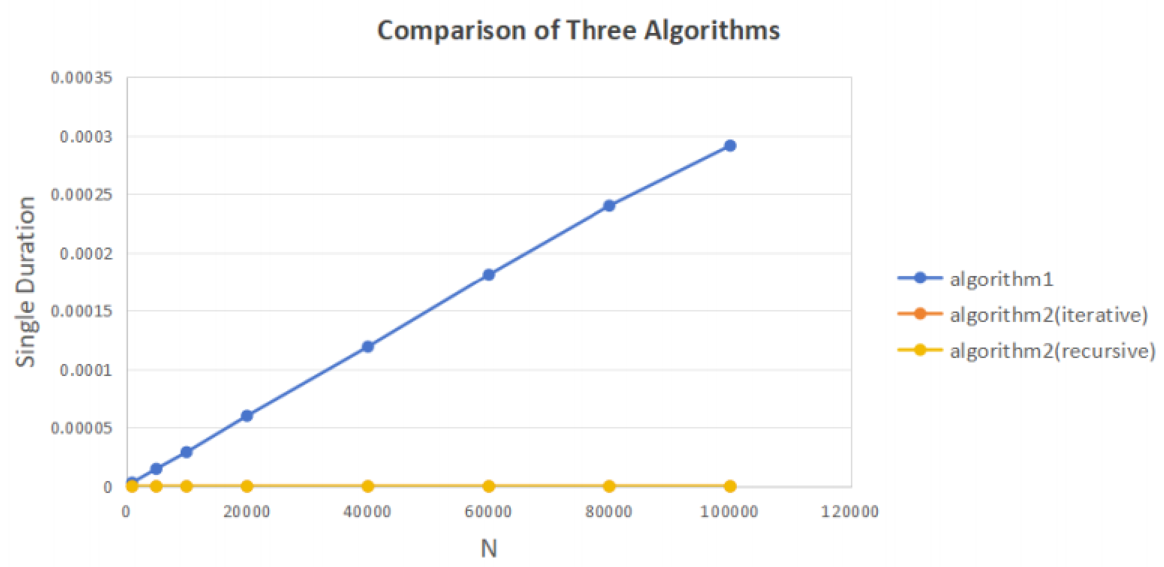
Chapter 3: Testing Results

	N	1000	5000	10000	20000
Algorithm1	Iterations(K)	10000	10000	1000	1000
	Ticks	32.000000	149.750000	29.250000	60.250000
	Total Durations(sec)	0.032000	0.149750	0.029250	0.060250
	Single Duration(sec)	0.0000032000	0.0000149750	0.0000292500	0.0000602500
Algorithm 2 (iterative version)	Iterations(K)	1000000	1000000	1000000	1000000
	Ticks	40.500000	62.250000	64.500000	69.500000
	Total Durations(sec)	0.040500	0.062250	0.064500	0.069500
	Single Duration(sec)	0.0000000405	0.0000000623	0.0000000645	0.0000000695
Algorithm 2 (recursive version)	Iterations(K)	1000000	1000000	1000000	1000000
	Ticks	32.250000	42.500000	46.000000	53.000000
	Total Durations(sec)	0.032250	0.042500	0.046000	0.053000
	Single Duration(sec)	0.0000000323	0.0000000425	0.0000000460	0.0000000530

	N	**40000**	**60000**	**80000**	**100000**
Algorithm1	Iterations(K)	1000	1000	1000	1000
	Ticks	119.250000	181.000000	240.000000	291.250000
	Total Durations(sec)	0.119250	0.181000	0.240000	0.291250
	Single Duration(sec)	0.0001192500	0.0001810000	0.0002400000	0.0002912500
Algorithm 2 (iterative version)	Iterations(K)	1000000	1000000	1000000	1000000
	Ticks	82.750000	76.750000	86.250000	64.250000
	Total Durations(sec)	0.082750	0.076750	0.086250	0.064250
	Single Duration(sec)	0.0000000828	0.0000000767	0.0000000863	0.0000000642
Algorithm 2 (recursive version)	Iterations(K)	1000000	1000000	1000000	1000000
	Ticks	54.750000	57.500000	60.250000	61.000000
	Total Durations(sec)	0.054750	0.057500	0.060250	0.061000
	Single Duration(sec)	0.0000000547	0.0000000575	0.0000000603	0.0000000610

Chapter 4: Analysis and Comments

- ## Runtime Comparison Chart



From the graph, it can be observed that when N is significantly large, Algorithm 1 takes considerably more time compared to the two variations of Algorithm 2. Moreover, there is not a substantial difference in terms of time overhead between the iterative and recursive versions of Algorithm 2, with the recursive version incurring slightly less time overhead. The reasons for the above-mentioned phenomenon can be understood from the perspective of time complexity. For algorithm 1, the loop will execute N-1 times, so obviously its time complexity is O(N). For algorithm 2(both iterative and recursive version), the loop will execute O(log N) times at most, so obviously its time complexity is O(log N) . When N is large, an algorithm with a time complexity of O(N) will be significantly slower than an algorithm with a time complexity of O(log N).

- ## Time complexity and Space complexity analysis

> **Algorithm 1**
>
> - Time Complexity: This algorithm uses a simple loop that iterates `n` times. Therefore, the time complexity is O(n).
> - Space Complexity: This algorithm uses only one additional variable, `result`, so the space complexity is O(1).

> **Algorithm 2 (Iterative Version)**
>
> - Time Complexity: This algorithm converts `n` to its binary representation and records each bit's information, then calculates based on the parity of each binary digit. The most time-consuming part is converting `n` to binary, which takes O(log n) operations. The subsequent loop iterations also take O(log n). Therefore, the overall time complexity is O(log n).
> - Space Complexity: This algorithm uses an array `data` of size 64 to store binary bit information, making the space complexity O(1) because the array size is a constant.

> **Algorithm 2 (Recursiveq Version)**
>
> - Time Complexity: This algorithm uses recursion to calculate the result. At each recursive step, the problem size is reduced by half because it divides `n` by 2. So, the depth of recursion is O(log n). Each recursive step performs constant work. Therefore, the overall time complexity is O(log n).
> - Space Complexity: Recursive calls require storing the state on the call stack, so the space complexity depends on the depth of recursion, which is O(log n).

Appendix: Source Code (in C)

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

// Global variables
clock_t start1,start2,start3,stop1,stop2,stop3;
double duration1,duration2,duration3;// Record execution time (sec)
double ticks1,ticks2,ticks3;
bool is_exit = false; // Whether to exit the main menu
bool is_exit_update = false; // Whether to exit the update menu for automation
double X = 1.0001;
long k1[8],k2[8],k3[8]; // Corresponding K values for different N values
long N[8] = {1000,5000,10000,20000,40000,60000,80000,100000};
double total_duration1[8],total_duration2[8],total_duration3[8]; // Accumulated time (after amplifying the loop)
double total_ticks1[8],total_ticks2[8],total_ticks3[8]; // Accumulated ticks
double total_average_ticks1[8],total_average_ticks2[8],total_average_ticks3[8]; // Used to output averages
double total_average_duration1[8],total_average_duration2[8],total_average_duration3[8]; // Used to output averages
double actual_duration1[8],actual_duration2[8],actual_duration3[8]; // Output actual time
int number = 0; // Record how many times automation has been performed

// Function declarations
double PowOfAlgorithm1(double x,int n); // Algorithm 1
double POWOfIterativeAlgorithm2(double x,int n); // Algorithm 2(iterative version)
double PowOfRecursiveAlgorithm2(double x,int n); // Algorithm 2(recursive version)
void ManualShowTest(); // Manual testing (for user input option 1)
void AutoShowTest(); // Automated testing (for user input option 2)
void UpdateGlobalValueOfK(); // Modify configuration items for automation testing (for user input option 3)
void ShowAverage(); // Output averages based on automation testing results and counts
void ShowMenu1(); // Print a more aesthetically pleasing main menu
void ShowMenu2(); // Print a more aesthetically pleasing update menu for automation

int main(void){
    // Initialize global variables
    for(int i=0;i<2;i++){
        k1[i]=10000;
    }
    for(int i=2;i<8;i++){
        k1[i]=1000;
    }
    for(int i=0;i<8;i++){
        k2[i]=1000000;
        k3[i]=1000000;
    }


    // Loop waiting for user input
    while(!is_exit){
        int op; // Set test mode
        ShowMenu1(); // Call a more aesthetically pleasing menu interface
        scanf("%d",&op);
        getchar();
        printf("\n");
        switch(op){
            case 1:
                // Manual testing
                ManualShowTest();
                break;
            case 2:
                // Automated testing
                AutoShowTest();
                break;
            case 3:
                // Modify configuration items for automation testing
                UpdateGlobalValueOfK();
                break;
            case 4:
                // Output averages based on automation testing results and counts
                ShowAverage();
                break;
            case 5:
                // Exit
                is_exit = true;
                break;
            default:
                printf("请输入正确的数字！\n");
                break;
        }
    }

    return 0;
}
```

```
//Algorithm 1
double PowOfAlgorithm1(double x,int n){
    int i;
    double result=1;    // Store the result
    for(i=0;i<n;i++){
        result=result*x;
    }
    return result;
}

//Algorithm 2(iterative version)
double POWOfIterativeAlgorithm2(double x,int n){
    int data[64],i,j;// The data array is used to record the parity after each division
    i=0;// Initialize i for subsequent loop traversal
    double result=x;// Store the result
    if(n==0){
        return 1;// Exclude the case where the exponent is 0
    }
    while(n!=1){
        if(n%2==1){
            data[i]=1;//1:odd
            i++;
        }else{
            data[i]=0;
            i++;//0:even
        }
        n=n>>1;// Divide n by 2
    }
    for(j=i-1;j>=0;j--){
        if(data[j]){
            result=result*result*x;// If data[j] is 1, it means the exponent is odd
        }else{
            result=result*result;// If data[j] is 0, it means the exponent is even
        }
    }
    return result;
}

//Algorithm 2(recursive version)
double PowOfRecursiveAlgorithm2(double x,int n){
    if(n==0){
        return 1;// N == 0, return 1
    }else if(n==1){
        return x;// N == 1, return x
    }else if(n%2==0){
        return PowOfRecursiveAlgorithm2(x*x,n/2);
    }else{
        return PowOfRecursiveAlgorithm2(x*x,(n-1)/2)*x;
    }
}

// Manual input to obtain test results
```

```c
void  ManualShowTest(){
    double X,result1,result2,result3;
    int N,k,K1,K2,K3;
    printf("请输入X和N:");
    scanf("%lf%d",&X,&N);   // Read X and N
    getchar();// Read the newline character
    printf("请输入Iterations K1 K2 K3:");// Read the number of iterations K1, K2, and K3
    scanf("%d%d%d",&K1,&K2,&K3);
    printf("\n");
    printf("Result      Ticks       Total Times(sec)\n");// Output format
    //Algorithm 1
    start1=clock();//start at the beginning of the function call
    for(k=0;k<K1;k++){
        result1=PowOfAlgorithm1(X,N);// Execute Algorithm 1, repeat K1 times
    }
    stop1=clock();//stop at the end of the function call
    ticks1=stop1-start1;// Calculate ticks
    duration1=((double)(stop1-start1))/CLK_TCK;// Calculate time
    printf("%f\t%f\t%f\t\n",result1,ticks1,duration1);
    printf("-----------------------------------------------\n");// Print separator

    //Algorithm 2(iterative version)
    start2=clock();//start at the beginning of the function call
    for(k=0;k<K2;k++){
        result2=POWOfIterativeAlgorithm2(X,N);// ExecuteAlgorithm 2(iterative version)，repeat K2 times
    }
    stop2=clock();//stop at the end of the function call
    ticks2=stop2-start2;// Calculate ticks
    duration2=((double)(stop2-start2))/CLK_TCK;// Calculate time
    printf("%f\t%f\t%f\t\n",result2,ticks2,duration2);
    printf("-----------------------------------------------\n");// Print separator

    //Algorithm 2(recursive version)
    start3=clock();//start at the beginning of the function call
    for(k=0;k<K3;k++){
        result3=PowOfRecursiveAlgorithm2(X,N);// ExecuteAlgorithm 2(recursive version)，repeat K3 times
    }
    stop3=clock();//stop at the end of the function call
    ticks3=stop3-start3;// CalculateTicks
    duration3=((double)(stop3-start3))/CLK_TCK;// Calculate time
    printf("%f\t%f\t%f\t\n",result3,ticks3,duration3);
}

// Automated testing, automatically give results based on recommended K values
void AutoShowTest(){
    number++; // Increment number each time automated testing is called
    double result1,result2,result3;
    for(int i=0;i<8;i++){
        //Algorithm 1
        start1=clock();//start at the beginning of the function call
        for(int k=0;k<k1[i];k++){
            result1=PowOfAlgorithm1(X,N[i]);// Execute Algorithm 1, repeat K1 times
        }
        stop1=clock();//stop at the end of the function call
        ticks1=stop1-start1;// Calculate ticks
        duration1=((double)(stop1-start1))/CLK_TCK;// Calculate time
        total_ticks1[i]+=ticks1;
        total_duration1[i]+= duration1; // Accumulate time
        printf("( X:%lf , N:%ld )\n",X,N[i]);
        printf("-----------------------------------------------\n");// Print separator
        printf("Result      Ticks       Total Times(sec)\n");
        printf("-----------------------------------------------\n");
        printf("%f\t%f\t%f\t\n",result1,ticks1,duration1);
        printf("-----------------------------------------------\n");

        //Algorithm 2(iterative version)
        start2=clock();//start at the beginning of the function call
        for(int k=0;k<k2[i];k++){
            result2=POWOfIterativeAlgorithm2(X,N[i]);
        }
        stop2=clock();//stop at the end of the function call
        ticks2=stop2-start2;
        duration2=((double)(stop2-start2))/CLK_TCK;
        total_ticks2[i]+=ticks2;
        total_duration2[i]+= duration2;
        printf("%f\t%f\t%f\t\n",result2,ticks2,duration2);
        printf("-----------------------------------------------\n");

        //Algorithm 2(recursive version)
        start3=clock();//start at the beginning of the function call
        for(int k=0;k<k3[i];k++){
            result3=PowOfRecursiveAlgorithm2(X,N[i]);
        }
        stop3=clock();//stop at the end of the function call
        ticks3=stop3-start3;
        duration3=((double)(stop3-start3))/CLK_TCK;
        total_ticks3[i]+=ticks3;
        total_duration3[i]+= duration3;
        printf("%f\t%f\t%f\t\n",result3,ticks3,duration3);
        printf("\n");
    }
}

void UpdateGlobalValueOfK(){
    is_exit_update = false; // Check if exit
    bool is_valid = false; // Check if user input is valid
    int op;
    int index;
    while(!is_exit_update){
        ShowMenu2(); // Call a more aesthetically pleasing menu interface
        scanf("%d",&op);
```

```c
        getchar();
        printf("\n");
        is_valid = false;
        switch(op){
            case 1:
                while(!is_valid){
                    printf("\n请输入您要修改的K值下标索引（从0-7分别代表1000/5000/10000/.../100000）：") ;
                    scanf("%d",&index);
                    getchar();
                    printf("\n");
                    if(index<0||index>7){
                        is_valid = false;
                        printf("请输入有效的下标索引\n");
                    }else{
                        printf("\n请输入您要修改成的值:");
                        scanf("%d",&k1[index]);
                        printf("\n");
                        is_valid = true;
                    }
                }
                break;
            case 2:
                while(!is_valid){
                    printf("\n请输入您要修改的K值下标索引（从0-7分别代表1000/5000/10000/.../100000）：") ;
                    scanf("%d",&index);
                    getchar();
                    printf("\n");
                    if(index<0||index>7){
                        is_valid = false;
                        printf("请输入有效的下标索引\n");
                    }else{
                        printf("\n请输入您要修改成的值:");
                        scanf("%d",&k2[index]);
                        printf("\n");
                        is_valid = true;
                    }
                }
                break;
            case 3:
                while(!is_valid){
                    printf("\n请输入您要修改的K值下标索引（从0-7分别代表1000/5000/10000/.../100000）：") ;
                    scanf("%d",&index);
                    getchar();
                    printf("\n");
                    if(index<0||index>7){
                        is_valid = false;
                        printf("请输入有效的下标索引\n");
                    }else{
                        printf("\n请输入您要修改成的值:");
                        scanf("%d",&k3[index]);
                        printf("\n");
                        is_valid = true;
                    }
                }
                break;
            case 4:
                printf("请输入您要修改的X的值：");
                scanf("%lf",&X);
                getchar();
                printf("\n");
                break;
            case 5:
                is_exit_update = true;
                break;
            default:
                printf("请输入正确的数字\n");
                break;
        }
    }

}

void ShowAverage(){

    for(int i=0;i<8;i++){
        total_average_duration1[i]=total_duration1[i]*1.0/number;
        total_average_duration2[i]=total_duration2[i]*1.0/number;
        total_average_duration3[i]=total_duration3[i]*1.0/number;
        total_average_ticks1[i]=total_ticks1[i]*1.0/number;
        total_average_ticks2[i]=total_ticks2[i]*1.0/number;
        total_average_ticks3[i]=total_ticks3[i]*1.0/number;
        actual_duration1[i]=total_average_duration1[i]*1.0/k1[i];
        actual_duration2[i]=total_average_duration2[i]*1.0/k2[i];
        actual_duration3[i]=total_average_duration3[i]*1.0/k3[i];
        printf("( X:%lf , N:%ld )\n",X,N[i]);
        printf("-----------------------------------------------------------------------\n");
        printf("K\t\tTicks\t\tTotal Durations(sec)\tSingle Duration\n");
        printf("-----------------------------------------------------------------------\n");
        printf("%ld\t\t%f\t%f\t\t%.10f\t\t\n",k1[i],total_average_ticks1[i],total_average_duration1[i],actual_duration1[i]);
        printf("-----------------------------------------------------------------------\n");
        printf("%ld\t\t%f\t%f\t\t%.10f\t\t\n",k2[i],total_average_ticks2[i],total_average_duration2[i],actual_duration2[i]);
        printf("-----------------------------------------------------------------------\n");
        printf("%ld\t\t%f\t%f\t\t%.10f\t\t\n",k3[i],total_average_ticks3[i],total_average_duration3[i],actual_duration3[i]);
        printf("\n");
    }
}

```

```c
void ShowMenu1(){
    // Print title
    printf( "+----------------------------------------------------------------+\n");
    printf( "| 选  项 |          请选择输入您的选择（1~5的数字）          |\n");
    printf( "+----------------------------------------------------------------+\n");
    // Print menu content
    printf( "|  [1]   | 手动测试：   手动输入X/N/K进行测试                    |\n");
    printf( "|  [2]   | 自动化测试：将按照推荐配置的X/N/K进行自动化测试         |\n");
    printf( "|  [3]   | 修改配置：   修改自动化测试默认配置的X/N/K的值          |\n");
    printf( "|  [4]   | 输出平均数：根据自动化测试的结果和总次数来输出平均数   |\n");
    printf( "|  [5]   | 退出：       退出程序                                 |\n");
    printf( "+----------------------------------------------------------------+\n");
    printf("\n");
}

void ShowMenu2(){
    // Print title
    printf( "+----------------------------------------------------------------+\n");
    printf( "| 选  项 |          请选择输入您的选择（1~5的数字）          |\n");
    printf( "+----------------------------------------------------------------+\n");
    // Print menu content
    printf( "|  [1]   | (K for algorithm1):           修改algorithm1中的K值     |\n");
    printf( "|  [2]   | (K for algorithm2(iterative)): 修改algorithm2中的K值     |\n");
    printf( "|  [3]   | (K for algorithm(recursive)):  修改algorithm2中的K值     |\n");
    printf( "|  [4]   | X:          修改X的值                                    |\n");
    printf( "|  [5]   | Exit:       退出选择                                    |\n");
    printf( "+----------------------------------------------------------------+\n");
    printf("\n");
}
```