

# 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

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

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

- pseudo-code of Algorithm2 (iterative version)

```
1 function PowOfIterativeAlgorithm2(x, n):
2     data[64]
3     i = 0
4     result = x
5     if n == 0:
6         return 1
7     while n != 1:
8         if n is odd:
9             data[i] = 1
10            i = i + 1
11        else:
12            data[i] = 0
13            i = i + 1
14        n = n >> 1
15    for j from i-1 down to 0:
16        if data[j] == 1:
17            result = result * result * x
18        else:
19            result = result * result
20    return result
21
```

(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)**

```
1 function PowOfRecursiveAlgorithm2(x, n):
2     if n == 0:
3         return 1
4     else if n == 1:
5         return x
6     else if n is even:
7         return PowOfRecursiveAlgorithm2(x * x, n / 2)
8     else:
9         return PowOfRecursiveAlgorithm2(x * x, (n - 1) / 2) * x
10
```

(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**

```
1 // Initialize global variables
2 for(int i=0;i<2;i++){
3     k1[i]=10000;
4 }
5 for(int i=2;i<8;i++){
6     k1[i]=1000;
7 }
8 for(int i=0;i<8;i++){
9     k2[i]=1000000;
10    k3[i]=1000000;
11 }
```

- **Second Part**

```
1 // Loop waiting for user input
2 while(!is_exit){
3     int op; // Set test mode
4     ShowMenu1(); // Call a more aesthetically pleasing menu
interface
```

```

5         scanf("%d",&op);
6         getchar();
7         printf("\n");
8         switch(op){
9             case 1:
10                // Manual testing
11                ManualShowTest();
12                break;
13            case 2:
14                // Automated testing
15                AutoShowTest();
16                break;
17            case 3:
18                // Modify configuration items for automation testing
19                UpdateGlobalValueOfK();
20                break;
21            case 4:
22                // Output averages based on automation testing results
and counts
23                ShowAverage();
24                break;
25            case 5:
26                // Exit
27                is_exit = true;
28                break;
29            default:
30                printf("请输入正确的数字! \n");
31                break;
32        }
33    }

```

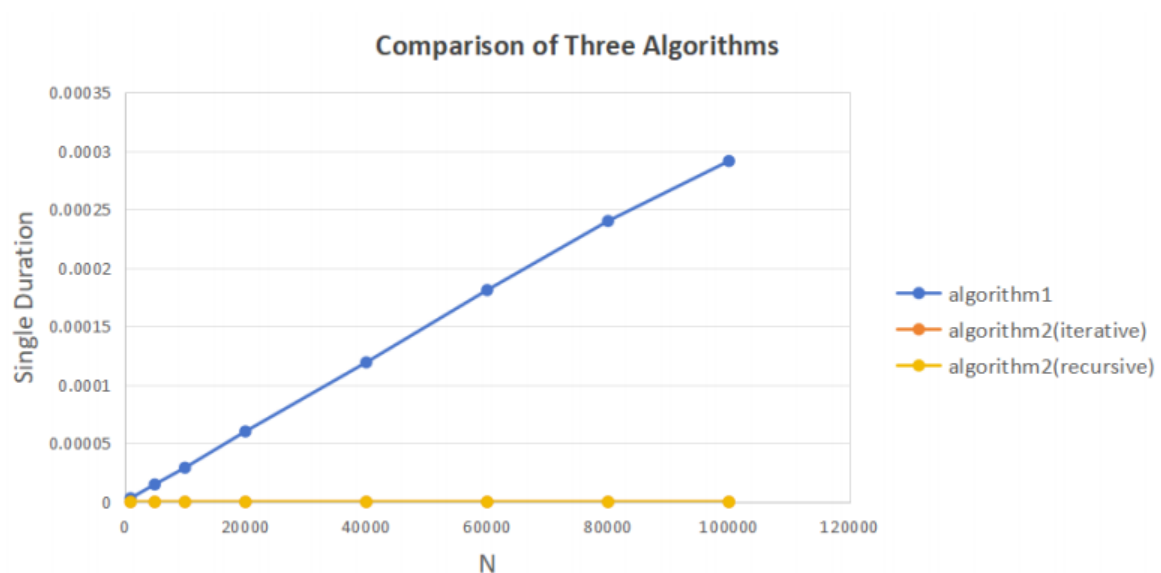
## Chapter 3: Testing Results

	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)

```
1  #include<stdio.h>
2  #include<time.h>
3
4  // Global variables
5  clock_t start1,start2,start3,stop1,stop2,stop3;
6  double duration1,duration2,duration3;// Record execution time (sec)
7  double ticks1,ticks2,ticks3;
8  bool is_exit = false; // whether to exit the main menu
9  bool is_exit_update = false; // whether to exit the update menu for
  automation
10 double x = 1.0001;
11 long k1[8],k2[8],k3[8]; // Corresponding k values for different N values
12 long N[8] = {1000,5000,10000,20000,40000,60000,80000,100000};
13 double total_duration1[8],total_duration2[8],total_duration3[8]; //
  Accumulated time (after amplifying the loop)
14 double total_ticks1[8],total_ticks2[8],total_ticks3[8]; // Accumulated
  ticks
15 double
  total_average_ticks1[8],total_average_ticks2[8],total_average_ticks3[8]; //
  Used to output averages
16 double
  total_average_duration1[8],total_average_duration2[8],total_average_duratio
  n3[8]; // Used to output averages
17 double actual_duration1[8],actual_duration2[8],actual_duration3[8]; //
  Output actual time
18 int number = 0; // Record how many times automation has been performed
19
20 // Function declarations
21 double PowOfAlgorithm1(double x,int n); // Algorithm 1
22 double POWOfIterativeAlgorithm2(double x,int n); // Algorithm 2(iterative
  version)
23 double PowOfRecursiveAlgorithm2(double x,int n); // Algorithm 2(recursive
  version)
24 void ManualShowTest(); // Manual testing (for user input option 1)
25 void AutoShowTest(); // Automated testing (for user input option 2)
26 void UpdateGlobalValueOfk(); // Modify configuration items for automation
  testing (for user input option 3)
27 void ShowAverage(); // Output averages based on automation testing results
  and counts
28 void ShowMenu1(); // Print a more aesthetically pleasing main menu
```



```

29 void ShowMenu2(); // Print a more aesthetically pleasing update menu for
    automation
30
31 int main(void){
32     // Initialize global variables
33     for(int i=0;i<2;i++){
34         k1[i]=10000;
35     }
36     for(int i=2;i<8;i++){
37         k1[i]=1000;
38     }
39     for(int i=0;i<8;i++){
40         k2[i]=1000000;
41         k3[i]=1000000;
42     }
43
44
45     // Loop waiting for user input
46     while(!is_exit){
47         int op; // Set test mode
48         ShowMenu1(); // Call a more aesthetically pleasing menu interface
49         scanf("%d",&op);
50         getchar();
51         printf("\n");
52         switch(op){
53             case 1:
54                 // Manual testing
55                 ManualShowTest();
56                 break;
57             case 2:
58                 // Automated testing
59                 AutoShowTest();
60                 break;
61             case 3:
62                 // Modify configuration items for automation testing
63                 UpdateGlobalValueOfK();
64                 break;
65             case 4:
66                 // Output averages based on automation testing results and
counts
67                 ShowAverage();
68                 break;
69             case 5:
70                 // Exit
71                 is_exit = true;
72                 break;
73             default:
74                 printf("请输入正确的数字! \n");
75                 break;
76         }
77     }
78
79     return 0;
80 }
81

```

```

82 //Algorithm 1
83 double PowOfAlgorithm1(double x,int n){
84     int i;
85     double result=1;    // Store the result
86     for(i=0;i<n;i++){
87         result=result*x;
88     }
89     return result;
90 }
91
92 //Algorithm 2(iterative version)
93 double POWofIterativeAlgorithm2(double x,int n){
94     int data[64],i,j;// The data array is used to record the parity after
each division
95     i=0;// Initialize i for subsequent loop traversal
96     double result=x;// Store the result
97     if(n==0){
98         return 1;// Exclude the case where the exponent is 0
99     }
100     while(n!=1){
101         if(n%2==1){
102             data[i]=1;//1:odd
103             i++;
104         }else{
105             data[i]=0;
106             i++;//0:even
107         }
108         n=n>>1;// Divide n by 2
109     }
110     for(j=i-1;j>=0;j--){
111         if(data[j]){
112             result=result*result*x;// If data[j] is 1, it means the
exponent is odd
113         }else{
114             result=result*result;// If data[j] is 0, it means the exponent
is even
115         }
116     }
117     return result;
118 }
119
120 //Algorithm 2(recursive version)
121 double PowOfRecursiveAlgorithm2(double x,int n){
122     if(n==0){
123         return 1;// N == 0, return 1
124     }else if(n==1){
125         return x;// N == 1, return x
126     }else if(n%2==0){
127         return PowOfRecursiveAlgorithm2(x*x,n/2);
128     }else{
129         return PowOfRecursiveAlgorithm2(x*x,(n-1)/2)*x;
130     }
131 }
132
133 // Manual input to obtain test results

```

```

134 void ManualShowTest(){
135     double x,result1,result2,result3;
136     int N,k,k1,k2,k3;
137     printf("请输入X和N:");
138     scanf("%lf%d",&x,&N);    // Read X and N
139     getchar();// Read the newline character
140     printf("请输入Iterations K1 K2 K3:");// Read the number of iterations
    K1, K2, and K3
141     scanf("%d%d%d",&k1,&k2,&k3);
142     printf("\n");
143     printf("Result      Ticks      Total Times(sec)\n");// Output format
144     //Algorithm 1
145     start1=clock();//start at the beginning of the function call
146     for(k=0;k<K1;k++){
147         result1=PowOfAlgorithm1(X,N);// Execute Algorithm 1, repeat K1
    times
148     }
149     stop1=clock();//stop at the end of the function call
150     ticks1=stop1-start1;// Calculate ticks
151     duration1=((double)(stop1-start1))/CLK_TCK;// Calculate time
152     printf("%f\t%f\t%f\t\n",result1,ticks1,duration1);
153     printf("-----\n");// Print
    separator
154
155     //Algorithm 2(iterative version)
156     start2=clock();//start at the beginning of the function call
157     for(k=0;k<K2;k++){
158         result2=POWOfIterativeAlgorithm2(X,N);// ExecuteAlgorithm
    2(iterative version), repeat K2 times
159     }
160     stop2=clock();//stop at the end of the function call
161     ticks2=stop2-start2;// Calculate ticks
162     duration2=((double)(stop2-start2))/CLK_TCK;// Calculate time
163     printf("%f\t%f\t%f\t\n",result2,ticks2,duration2);
164     printf("-----\n");// Print
    separator
165
166     //Algorithm 2(recursive version)
167     start3=clock();//start at the beginning of the function call
168     for(k=0;k<K3;k++){
169         result3=PowOfRecursiveAlgorithm2(X,N);// ExecuteAlgorithm
    2(recursive version), repeat K3 times
170     }
171     stop3=clock();//stop at the end of the function call
172     ticks3=stop3-start3;// CalculateTicks
173     duration3=((double)(stop3-start3))/CLK_TCK;// Calculate time
174     printf("%f\t%f\t%f\t\n",result3,ticks3,duration3);
175 }
176
177 // Automated testing, automatically give results based on recommended K
    values
178 void AutoShowTest(){
179     number++; // Increment number each time automated testing is called
180     double result1,result2,result3;
181     for(int i=0;i<8;i++){

```

```

182         //Algorithm 1
183         start1=clock();//start at the beginning of the function call
184         for(int k=0;k<k1[i];k++){
185             result1=PowOfAlgorithm1(X,N[i]); // Execute Algorithm 1, repeat
K1 times
186         }
187         stop1=clock();//stop at the end of the function call
188         ticks1=stop1-start1; // Calculate ticks
189         duration1=((double)(stop1-start1))/CLK_TCK; // Calculate time
190         total_ticks1[i]+=ticks1;
191         total_duration1[i]+= duration1; // Accumulate time
192         printf("( X:%lf , N:%ld )\n",X,N[i]);
193         printf("-----\n");//
Print separator
194         printf("Result      Ticks      Total Times(sec)\n");
195         printf("-----\n");
196         printf("%f\t%f\t%f\t\n",result1,ticks1,duration1);
197         printf("-----\n");
198
199         //Algorithm 2(iterative version)
200         start2=clock();//start at the beginning of the function call
201         for(int k=0;k<k2[i];k++){
202             result2=POWOfIterativeAlgorithm2(X,N[i]);
203         }
204         stop2=clock();//stop at the end of the function call
205         ticks2=stop2-start2;
206         duration2=((double)(stop2-start2))/CLK_TCK;
207         total_ticks2[i]+=ticks2;
208         total_duration2[i]+= duration2;
209         printf("%f\t%f\t%f\t\n",result2,ticks2,duration2);
210         printf("-----\n");
211
212         //Algorithm 2(recursive version)
213         start3=clock();//start at the beginning of the function call
214         for(int k=0;k<k3[i];k++){
215             result3=PowOfRecursiveAlgorithm2(X,N[i]);
216         }
217         stop3=clock();//stop at the end of the function call
218         ticks3=stop3-start3;
219         duration3=((double)(stop3-start3))/CLK_TCK;
220         total_ticks3[i]+=ticks3;
221         total_duration3[i]+= duration3;
222         printf("%f\t%f\t%f\t\n",result3,ticks3,duration3);
223         printf("\n");
224     }
225 }
226
227 void UpdateGlobalValueOfk(){
228     is_exit_update = false; // Check if exit
229     bool is_valid = false; // Check if user input is valid
230     int op;
231     int index;
232     while(!is_exit_update){
233         ShowMenu2(); // Call a more aesthetically pleasing menu interface
234         scanf("%d",&op);

```

```

235     getchar();
236     printf("\n");
237     is_valid = false;
238     switch(op){
239         case 1:
240             while(!is_valid){
241                 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）：");
242                 scanf("%d",&index);
243                 getchar();
244                 printf("\n");
245                 if(index<0||index>7){
246                     is_valid = false;
247                     printf("请输入有效的下标索引\n");
248                 }else{
249                     printf("\n请输入您要修改成的值:");
250                     scanf("%d",&k1[index]);
251                     printf("\n");
252                     is_valid = true;
253                 }
254             }
255             break;
256         case 2:
257             while(!is_valid){
258                 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）：");
259                 scanf("%d",&index);
260                 getchar();
261                 printf("\n");
262                 if(index<0||index>7){
263                     is_valid = false;
264                     printf("请输入有效的下标索引\n");
265                 }else{
266                     printf("\n请输入您要修改成的值:");
267                     scanf("%d",&k2[index]);
268                     printf("\n");
269                     is_valid = true;
270                 }
271             }
272             break;
273         case 3:
274             while(!is_valid){
275                 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）：");
276                 scanf("%d",&index);
277                 getchar();
278                 printf("\n");
279                 if(index<0||index>7){
280                     is_valid = false;
281                     printf("请输入有效的下标索引\n");
282                 }else{
283                     printf("\n请输入您要修改成的值:");
284                     scanf("%d",&k3[index]);
285                     printf("\n");
286                     is_valid = true;

```

```

287         }
288     }
289     break;
290     case 4:
291         printf("请输入您要修改的X的值: ");
292         scanf("%lf",&x);
293         getchar();
294         printf("\n");
295         break;
296     case 5:
297         is_exit_update = true;
298         break;
299     default:
300         printf("请输入正确的数字\n");
301         break;
302     }
303 }
304
305 }
306
307 void showAverage(){
308
309     for(int i=0;i<8;i++){
310         total_average_duration1[i]=total_duration1[i]*1.0/number;
311         total_average_duration2[i]=total_duration2[i]*1.0/number;
312         total_average_duration3[i]=total_duration3[i]*1.0/number;
313         total_average_ticks1[i]=total_ticks1[i]*1.0/number;
314         total_average_ticks2[i]=total_ticks2[i]*1.0/number;
315         total_average_ticks3[i]=total_ticks3[i]*1.0/number;
316         actual_duration1[i]=total_average_duration1[i]*1.0/k1[i];
317         actual_duration2[i]=total_average_duration2[i]*1.0/k2[i];
318         actual_duration3[i]=total_average_duration3[i]*1.0/k3[i];
319         printf("( X:%lf , N:%ld )\n",x,N[i]);
320         printf("-----\n");
321         printf("K\tTicks\tTotal Durations(sec)\tSingle Duration\n");
322         printf("-----\n");
323
324         printf("%ld\t%f\t%f\t\t%.10f\t\t\n",k1[i],total_average_ticks1[i],total_a
verage_duration1[i],actual_duration1[i]);
325         printf("-----\n");
326
327         printf("%ld\t%f\t%f\t\t%.10f\t\t\n",k2[i],total_average_ticks2[i],total_a
verage_duration2[i],actual_duration2[i]);
328         printf("-----\n");
329
330         printf("%ld\t%f\t%f\t\t%.10f\t\t\n",k3[i],total_average_ticks3[i],total_a
verage_duration3[i],actual_duration3[i]);
331         printf("\n");
332     }
333 }

```

```

332 void ShowMenu1(){
333     // Print title
334     printf( "+-----+
---+\n");
335     printf( "| 选 项 |           请选择输入您的选择（1~5的数字）
|\n");
336     printf( "+-----+
---+\n");
337     // Print menu content
338     printf( "|  [1]  | 手动测试：   手动输入X/N/K进行测试
|\n");
339     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");
340     printf( "|  [3]  | 修改配置：   修改自动化测试默认配置的X/N/K的值
|\n");
341     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");
342     printf( "|  [5]  | 退出：       退出程序
|\n");
343     printf( "+-----+
---+\n");
344     printf("\n");
345 }
346
347 void ShowMenu2(){
348     // Print title
349     printf( "+-----+
---+\n");
350     printf( "| 选 项 |           请选择输入您的选择（1~5的数字）
|\n");
351     printf( "+-----+
---+\n");
352     // Print menu content
353     printf( "|  [1]  | (K for algorithm1):           修改algorithm1中的K值
|\n");
354     printf( "|  [2]  | (K for algorithm2(iterative)): 修改algorithm2中的K值
|\n");
355     printf( "|  [3]  | (K for algorithm(recursive)):  修改algorithm2中的K值
|\n");
356     printf( "|  [4]  | x:           修改x的值
|\n");
357     printf( "|  [5]  | Exit:        退出选择
|\n");
358     printf( "+-----+
---+\n");
359     printf("\n");
360 }
361
362

```