Exponentiation by Squaring

刘思锐

Date: 2021-10-8

Chap.1-Introduction

How can we compute X^N, for example, 7^{10}?

Algorithm 1

7*7=49, 49*7=343, 343*7......

In this way, for X^N, we need $N-1$ multiplications.

Algorithm 2

To compute 7^{10}, we need 7^5. The square of 7^5 equals to 7^{10}.

To compute 7^5, we need 7^2. The square of 7^2 multiplying 7 equals to 7^5.

To compute 7^2, we need......

For X^N, if N is even,

$$X^N = X^{(N/2)} * X^{(N/2)}$$

And if N is odd,

$$X^N = X * X^{(N/2)} * X^{(N/2)}$$

In this way, we can compute X^N faster than Algorithm 1, owing to less multiplications we need. Algorithm 2 is also called **Exponentiation by squaring**.

These two algorithms will be completed in this project and Algorithm 2 will be completed in both iterative and recursive way. The performance of each algorithm will also be analysed.

Chap.2-Algorithm Specification

Algorithm 1

```
double result=x;
for(int i=0;i<N-1;i++){//compute X^N
result*=x;
}
```

Algorithm 1 repeat the same multiplication for *N*-1 times.

Algorithm 2 (iterative version)

```
double result=1;
double base=x;
while(N>0){//compute X^N
   if(N%2){
         result*=base;
   }
    base*=base;
    N/=2;
}
```

Base indicates *2^M* times of *X* starting with *M=1*. Every time *N%2 ==1*, which means a certain bit of N(bin) equals to 1, our result should multiply by the base now. For example, $\boldsymbol{2^{5(ten)} = 2^{101(bin)}}$ and $\boldsymbol{2^{101(bin)} = 2^{1(bin)} * 2^{100(bin)}}$. It's a little different from the giving formula in introduction but actually they're the same thing. The key is to convert the exponent N from decimal to binary.

Algorithm 3 (recursive version)

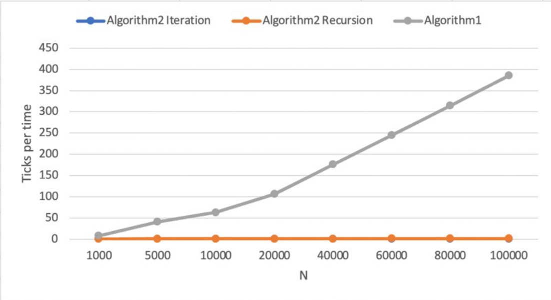
```
result=POW(x, N);

double POW (double x, int N)
{
```

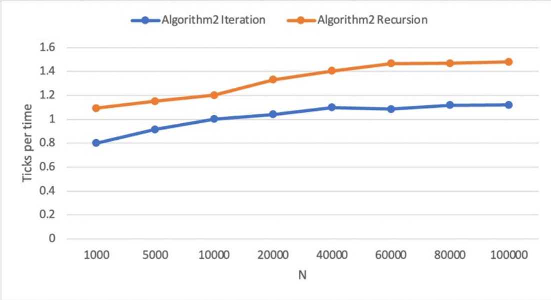
```
if(N==0) return 1;
    else if(N==1) return x;
    else return POW(x, N/2)*POW(x, N/2)*x;
}
```

It is the same as the formula given in Chap.1.

Chap.3-Testing result

	N	1000	5000	10000	20000	40000	60000	80000	100000
Algorithm 1	Iterations (K)	103	103	103	103	103	103	103	103
	Ticks	8431	40864	63337	106409	175959	244528	313993	384834
	Total Time (sec)	$8.43*10^{-3}$	$4.08*10^{-2}$	$6.33*10^{-2}$	$1.06*10^{-1}$	$1.76*10^{-1}$	$2.45*10^{-1}$	$3.14*10^{-1}$	$3.85*10^{-1}$
	Duration (sec per time)	$8.43*10^{-6}$	$4.08*10^{-5}$	$6.33*10^{-5}$	$1.06*10^{-4}$	$1.76*10^{-4}$	$2.45*10^{-4}$	$3.14*10^{-4}$	$3.85*10^{-4}$
Algorithm 2 (recursive version)	Iterations (K)	106	106	106	106	106	106	106	106
	Ticks	109271	114966	120081	133063	140495	146583	146815	147858
	Total Time (sec)	$1.09*10^{-1}$	$1.15*10^{-1}$	$1.20*10^{-1}$	$1.33*10^{-1}$	$1.40*10^{-1}$	$1.47*10^{-1}$	$1.47*10^{-1}$	$1.48*10^{-1}$
	Duration (sec per time)	$1.09*10^{-7}$	$1.15*10^{-7}$	$1.20*10^{-7}$	$1.33*10^{-7}$	$1.40*10^{-7}$	$1.47*10^{-7}$	$1.47*10^{-7}$	$1.48*10^{-7}$
Algorithm 3 (iterative version)	Iterations (K)	106	106	106	106	106	106	106	106
	Ticks	80050	91381	100120	103959	109851	108528	111702	111905
	Total Time (sec)	$8.01*10^{-2}$	$9.14*10^{-2}$	$1.00*10^{-1}$	$1.00*10^{-1}$	$1.10*10^{-1}$	$1.08*10^{-1}$	$1.11*10^{-1}$	$1.12*10^{-1}$
	Duration (sec per time)	$8.01*10^{-8}$	$9.14*10^{-8}$	$1.00*10^{-7}$	$1.00*10^{-7}$	$1.10*10^{-7}$	$1.08*10^{-7}$	$1.11*10^{-7}$	$1.12*10^{-7}$





Testing purpose: measure the time each program takes with the same X=0.0001 and different N ranging from 1000 to 100000 to compare the performance of algorithms.

Chap.4-Analysis and Comments

Algorithm 1 need *N-1* multiplications, so the time complexity is ***O(N)***. It needs no extra space to store figures, so the space complexity is ***O(1)***.

Denote the times of multiplication Algorithm 2 need as *T*. $2^T=N$, so the time complexity of algorithm 2 is ***$O(\log_2 N)$***. Iterative version need no extra space to store figures. Space complexity of iterative version is ***O(1)***. The recursion depth of recursive version is equal to the times of multiplications, so the space complexity is ***$O(\log_2 N)$***.

We can also tell that iteration is usually faster than recursion and taking up less space in an algorithm. It's a better choice in most cases.

Declaration

I hereby declare that all the works done in this project titled "Progect1" is of my independent effort.

Appendix-Source Code

Algorithm 1

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

clock_t start, stop;
double duration;

int main ()
{
    //initialize
double x=1.0001, result;
int N;
    printf("Input N:\n");
scanf("%d", &N);//input N

start=clock();
for(int j=0;j<1000;j++)//repeat 10^3 times
{
//re-initialize
result=1;
//compute x^N
for(int i=0;i<N;i++){
result*=x;
}
}
stop=clock();

duration=(double)(stop-start)/(double)CLOCKS_PER_SEC;//convert clock into seconds
printf("Ticks:%d Seconds:%f\n",stop-start , duration);//output result

return 0;
}
```

Algorithm 2 (Recursive Version)

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

double POW (double x, int N);

clock_t start, stop;
double duration;

```

```
int main ()
{
    //initialize
double x=1.0001, result, base;
int N, temp;
    printf("Input N:\n");
scanf("%d", &N);//input N

start=clock();
for(int i=0;i<1000000;i++){//repeat 10^6 times
//compute x^n by recursion
result=POW(x, N);
}
stop=clock();

duration=(double)(stop-start)/(double)CLOCKS_PER_SEC;//convert clock into seconds
printf("Ticks:%d Seconds:%f\n",stop-start , duration);//output result

return 0;
}

double POW (double x, int N)
{
double result;
//exit of recursion
if(N==0) result=1;
else if(N==1) result=x;
//continue
else{
result=POW(x, N/2);
result*=result;
if(N%2) result*=x;
}
//return
return result;
}
```

Algorithm 2 (Iterative Version)

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

clock_t start, stop;
double duration;
```

```c

int main ()
{
double x=1.0001, result, base;
int N, temp;
    printf("Input N:\n");
scanf("%d", &N);//input N

start=clock();
for(int i=0;i<1000000;i++){//repeate 10^6 times
//re-initialize
result=1;
base=x;
temp=N;
//compute x^N by iterative algorithm
while(temp>0){
              //detail of this part is given
             //above in Algorithm Specification
if(temp%2){
result*=base;
}
base*=base;
temp/=2;
}
}
stop=clock();

duration=(double)(stop-start)/(double)CLOCKS_PER_SEC;//convert clock into seconds
printf("Ticks:%d Seconds:%f\n",stop-start , duration);//output result

return 0;
}
```