**Normal Performance Measurement (POW)**

**Date: 2023-10-8**

### Chapter 1: Introduction

There are at least two different algorithms that can compute X^N for some positive integer N.

Algorithm 1 is to use N−1 multiplications.

Algorithm 2 works in the following way: if N is even, X^N=X^(N/2)×X^(N/2); and if N is odd, X^N =X^[(N−1)/2]×X^[(N−1)/2]×X.

Tasks are:

(1) Implement Algorithm 1 and an iterative version of Algorithm 2;

(2) Analyze the complexities of the two algorithms;

(3) Measure and compare 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.

### Chapter 2: Algorithm Specification

### Algorithm 1

{

Print: “Please enter exponent::;

Initialization variable:N,I,ttime,ticks,T;

Set T as K;

Input N;

Set x an 1.0001;

Loop the program for K time{

Start time;

Initialization variable res,which equal to x;

Print the res as result;

Stop time;

Calculate the duration from ticks;

Add up the ticks spent in each round to get the total ticks;

Add up the time spent in each round to get the total time;

}

Print K,total ticks and total time;

End the program;

}

### Algorithm 2-rescurion

{

Print: “Please enter exponent:”;

Initialization variable:N,I,ttime,ticks,T;

Set T as K;

Input N;

Set x an 1.0001;

Loop the program for K time{

Start time;

Determine the parity of N and call the corresponding recursive function;

If N is odd,then it will call the function ODD,which takes x and N:

{

If N<=1,then end the function with returning x;

If N>1,then subtract 1 from N, divide N-1 by two, and it get the new N;

Then call the EVEN function with x and the new N,square the result and multiply it by x,and return the final result;

}

If N is even,then it will call the function EVEN,which takes x and N:

{

If N<=1,then end the function with returning x;

If N>1, divide N by two, then it get the new N;

Then call the EVEN function with x and the new N,square the result and return the final result;

}

Print the result;

Stop time;

Calculate the duration from ticks;

Add up the ticks spent in each round to get the total ticks;

Add up the time spent in each round to get the total time;

}

Print K,total ticks and total time;

End the program;

}

### Algorithm 2-iteration

{

Print: “Please enter exponent:”;

Initialization variable:N,I,ttime,ticks,T;

Set T as K;

Input N;

Set x an 1.0001;

Loop the program for K time{

Start time;

Call the function POWER with N and x:

{

Initialization variable res as 1 to store the result;

Loop through the following operations for n greater than 0:

{

Determine the parity of N and call the corresponding recursive function,if N is odd,then multiply res by x;

Multiply x by x;

Divide N by 2 to get the new N;

}

}

Print the result;

Stop time;

Calculate the duration from ticks;

Add up the ticks spent in each round to get the total ticks;

Add up the time spent in each round to get the total time;

}

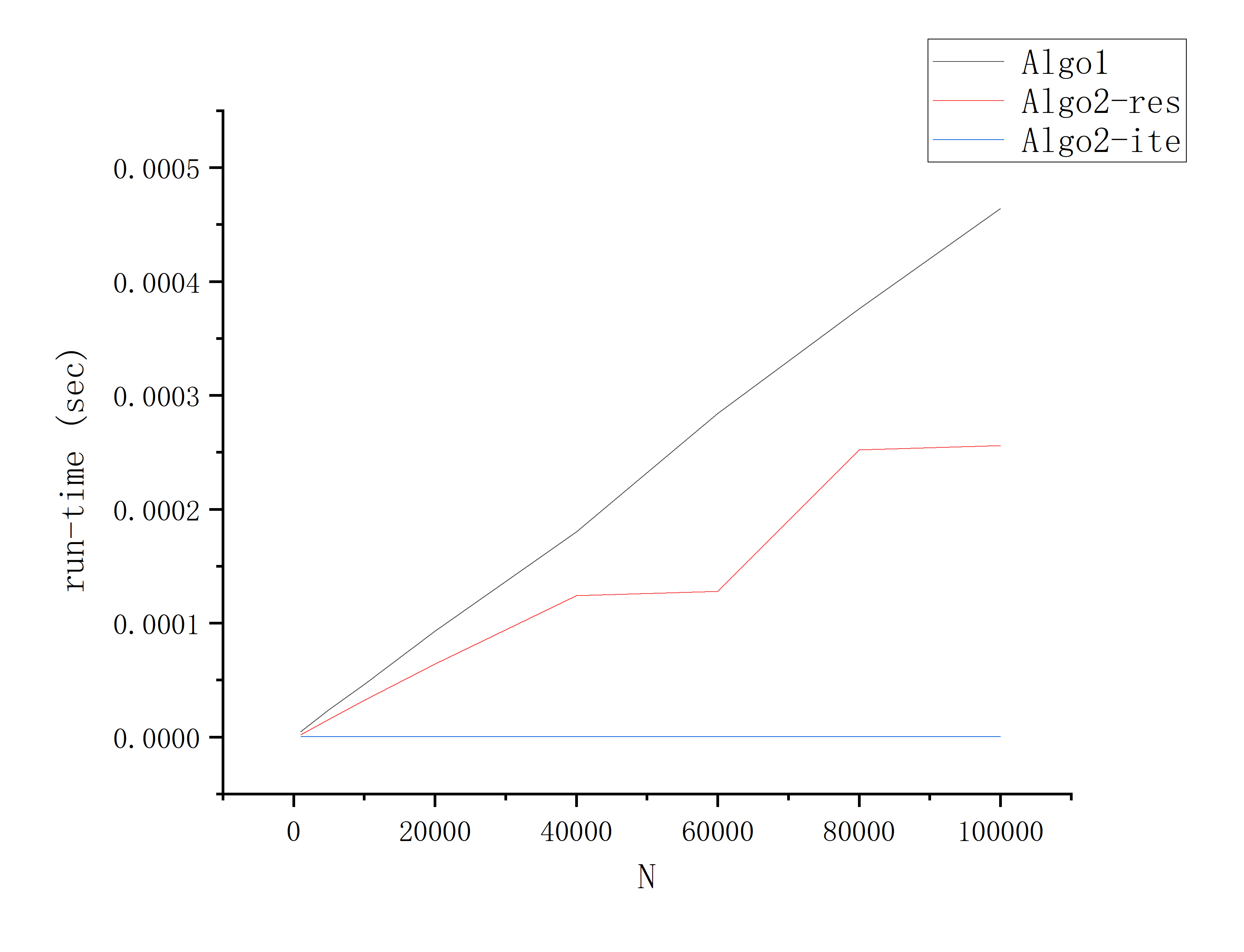
Print K,total ticks and total time;

End the program;

}

### Chapter 3: Testing Results

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | N | 1000 | 5000 | 10000 | 20000 | 40000 | 60000 | 80000 | 100000 |
| Algorithm 1 | Iterations(K) | 10000 | 1000 | 1000 | 1000 | 500 | 250 | 250 | 250 |
| Ticks | 46 | 24 | 46 | 93 | 90 | 71 | 96 | 116 |
| Total Time (sec) | 0.046 | 0.024 | 0.046 | 0.093 | 0.09 | 0.071 | 0.094 | 0.116 |
| Duration(sec) | 0.0000046 | 0.000024 | 0.000046 | 0.000093 | 0.00018 | 0.000284 | 0.000376 | 0.000464 |
| Algorithm 2-res | Iterations(K) | 10000 | 5000 | 1000 | 1000 | 500 | 500 | 250 | 250 |
| Ticks | 19 | 77 | 32 | 64 | 62 | 64 | 63 | 64 |
| Total Time (sec) | 0.019 | 0.077 | 0.032 | 0.064 | 0.062 | 0.064 | 0.063 | 0.064 |
| Duration(sec) | 0.0000019 | 0.0000154 | 0.000032 | 0.000064 | 0.000124 | 0.000128 | 0.000252 | 0.000256 |
| Algorithm 2-ite | Iterations(K) | 1000000 | 1000000 | 1000000 | 1000000 | 100000 | 1000000 | 1000000 | 1000000 |
| Ticks | 15 | 18 | 21 | 23 | 27 | 26 | 27 | 26 |
| Total Time (sec) | 0.015 | 0.018 | 0.021 | 0.023 | 0.027 | 0.026 | 0.027 | 0.026 |
| Duration(sec) | 0.000000015 | 0.000000018 | 0.000000021 | 0.000000023 | 0.00000027 | 0.000000026 | 0.000000027 | 0.000000026 |



The results are generally in line with expectations(all pass).

### Chapter 4: Analysis and Comments

Analysis of the time and space complexities of the algorithms.

Time complexities:

Algo1:O(N\*K)

Algo2-res: O(K\*logN)

Algo2-ite: O(K)

Space complexity:

Algo1:O(1)

Algo2-res: O(logN)

Algo2-ite:O(1)

Comments on further possible improvements:

Algo1:Use bitwise operations to check the parity of N in the while loop instead of using modulus operation, as bitwise operations are generally faster, but unfortunately I didn't know until after I finished the statistics, so I didn't change the code.

Algo2-res:Handle the exponent N with special cases, such as dividing it by 2 when N is even and squaring the base number x, to reduce the number of iterations and improve code speed and memory usage.

Algo2-ite: I don't have an idea about it at the moment.

### Appendix: Source Code (in C)

Appendix is in the code folder.

### Declaration

***I hereby declare that all the work done in this project titled "*** ***Normal Performance Measurement (POW)" is of my independent effort.***