Fundamentals of Data Structures

Laboratory Projects

Performance Measurement (POW)

周承扬 3190102371

Date: 2021-10-08

Chapter 1: Introduction

The project needs us to compare two algorithms that can compute X^N for some positive integer N.

Algorithm 1 is to use N−1 multiplications.

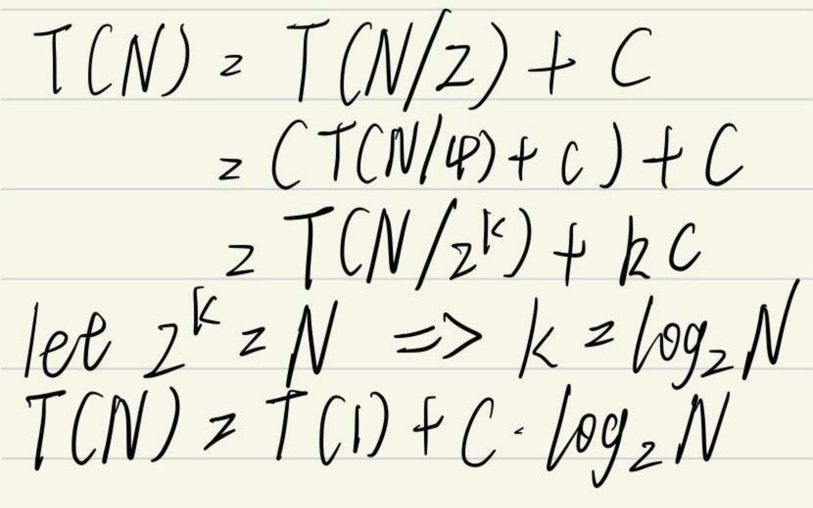
Algorithm 2 works in the following way: if N is even, X^N=X^N/2×X^N/2; and if N is odd, X^N=X^(N−1)/2×X^(N−1)/2×X. And for the algorithm there are iterative and recursive implementations to solve the problem.

As a nutshell we should analyze three algorithms in all.

Chapter 2: Algorithm Specification

Algorithm 1 is to use N−1 multiplications. So that we can easily know that the time complexity is O(N).

Algorithm 2 is 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. So that the time complexity is O(N). Prove as follows:



$$
\begin{aligned}
T(N) &= T(N/2) + C \\
&= (T(N/4) + c) + C \\
&= T(N/2^k) + kC
\end{aligned}
$$

$$\text{let } 2^k = N \Rightarrow k = \log_2 N$$

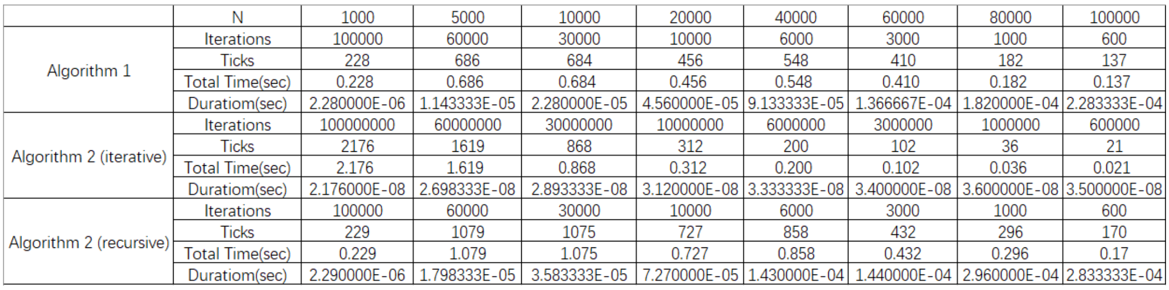
$$T(N) = T(1) + C \cdot \log_2 N$$

The proof is done.

So, theoretically algorithm 2 is better than Algorithm 1.

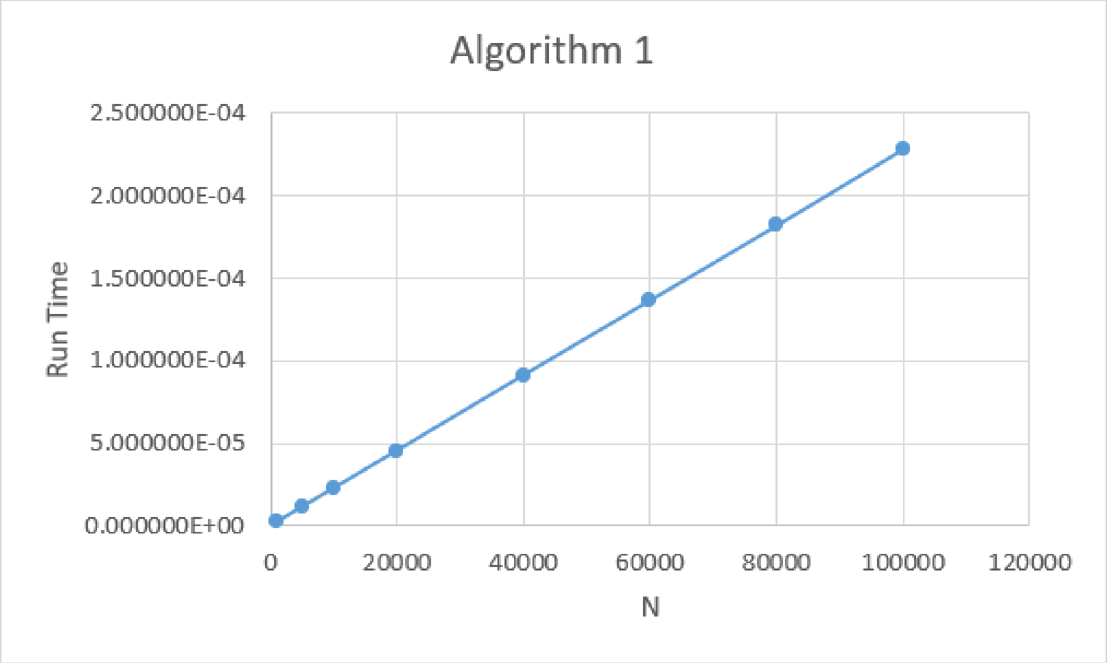
Chapter 3: Testing Results

The test results are shown below:

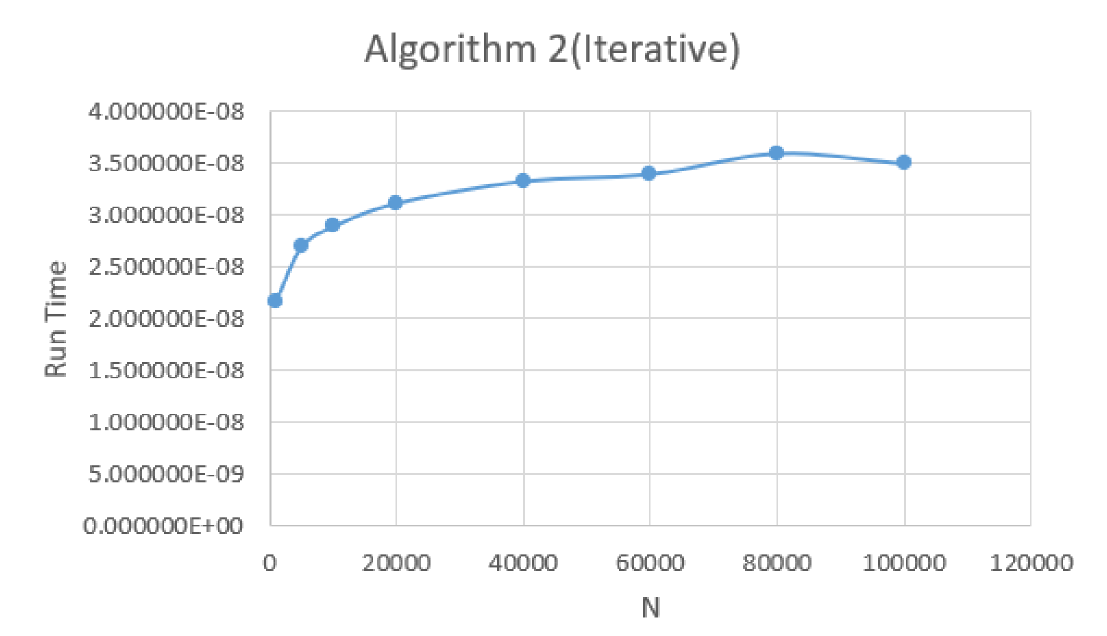


	N	1000	5000	10000	20000	40000	60000	80000	100000
Algorithm 1	Iterations	100000	60000	30000	10000	6000	3000	1000	600
	Ticks	228	686	684	456	548	410	182	137
	Total Time(sec)	0.228	0.686	0.684	0.456	0.548	0.410	0.182	0.137
	Duratiom(sec)	2.280000E-06	1.143333E-05	2.280000E-05	4.560000E-05	9.133333E-05	1.366667E-04	1.820000E-04	2.283333E-04
Algorithm 2 (iterative)	Iterations	100000000	60000000	30000000	10000000	6000000	3000000	1000000	600000
	Ticks	2176	1619	868	312	200	102	36	21
	Total Time(sec)	2.176	1.619	0.868	0.312	0.200	0.102	0.036	0.021
	Duratiom(sec)	2.176000E-08	2.698333E-08	2.893333E-08	3.120000E-08	3.333333E-08	3.400000E-08	3.600000E-08	3.500000E-08
Algorithm 2 (recursive)	Iterations	100000	60000	30000	10000	6000	3000	1000	600
	Ticks	229	1079	1075	727	858	432	296	170
	Total Time(sec)	0.229	1.079	1.075	0.727	0.858	0.432	0.296	0.17
	Duratiom(sec)	2.290000E-06	1.798333E-05	3.583333E-05	7.270000E-05	1.430000E-04	1.440000E-04	2.960000E-04	2.833333E-04

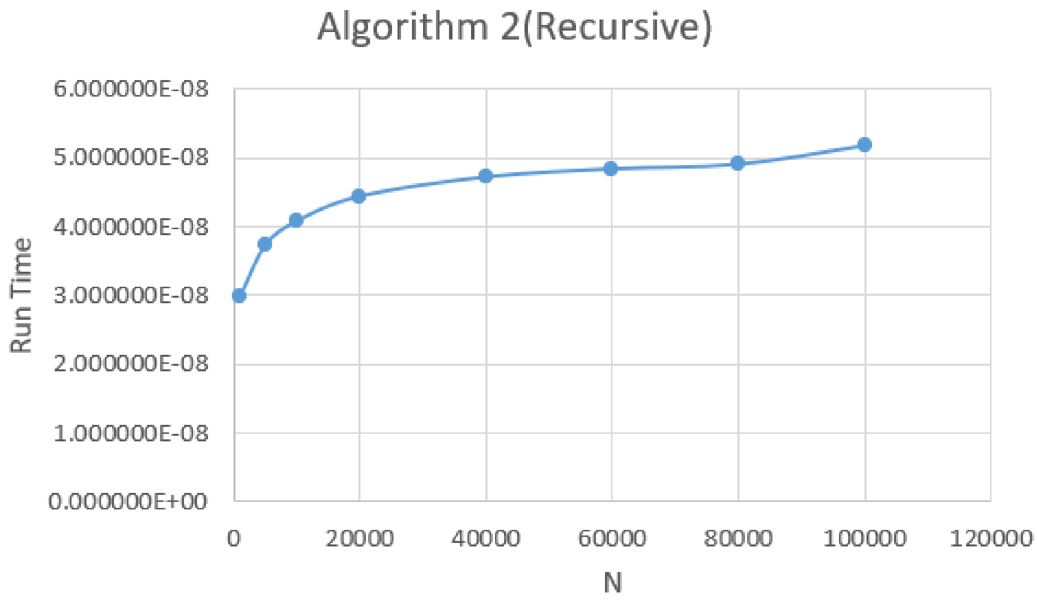
The run time of Algorithm 1:



The run time of Algorithm 2(Iterative):



The run time of Algorithm 2(Recursive):



Chapter 4: Analysis and Comments

Algorithm 1: time complexity is O(N) and space complexity is O(1).

Algorithm 2: time complexity is O(logN) and space complexity is O(1).

Combined with the above experimental results, algorithm 2 is better than algorithm 1.

Appendix: Source Code (in C)

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

//construct correlation function
double powNormal(double x, int n);
double powHardInterative(double x, int n);
double powHardRecursive(double x, int  n);

int main() {
    clock_t start, end;
    double tick[8];
    double total[8];
    double duration[8];

    double x = 1.0001;
    int n[] = {1000, 5000, 10000, 20000, 40000, 60000, 80000, 100000};
    int k[] = {100000, 60000, 30000, 10000, 6000, 3000, 1000, 600}; //k[i] is Iteration(K)
    int i, j;

    printf("N      ");
    for(i = 0; i < 8; i++) printf("%d      ", n[i]);
    printf("\n");

    for(i = 0; i < 8; i++) {
        start = clock();
        for(j = 0; j < k[i]; j++) {
            powNormal(x, n[i]);
        }
        end = clock();
        tick[i] = end - start; //Calculate the ticks, the same as below
        total[i] = tick[i]/CLK_TCK; //Calculate the total time, the same as below
        duration[i] = total[i]/k[i]; //Calculate the duration time, the same as below
    }
    printf("Algorithm 1: ");
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%d     ", k[i]);
    }
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%f  ", tick[i]);
    }
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%f  ", total[i]);
    }
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%e  ", duration[i]);
    }
    printf("\n"); //Output the required four columns of data respectively, the same as below

    for(i = 0; i < 8; i++) k[i] *= 1000; //Adjust the k[i] according to the algotithm and the project
    for(i = 0; i < 8; i++) {
        start = clock();
        for(j = 0; j < k[i]; j++) {
            powHardRecursive(x, n[i]);
        }
        end = clock();
        tick[i] = end - start;
        total[i] = tick[i]/CLK_TCK;
        duration[i] = total[i]/k[i];
    }
    printf("Algorithm 2(Recursive): ");
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%d     ", k[i]);
    }
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%f  ", tick[i]);
    }
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%f  ", total[i]);
    }
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%e  ", duration[i]);
    }
    printf("\n");

    for(i = 0; i < 8; i++) {
        start = clock();
        for(j = 0; j < k[i]; j++) {
            powHardInterative(x, n[i]);
        }
        end = clock();
        tick[i] = end - start;
        total[i] = tick[i]/CLK_TCK;
        duration[i] = total[i]/k[i];
    }
    printf("Algorithm 2(Iterative): ");
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%d     ", k[i]);
    }
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%f  ", tick[i]);
    }
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%f  ", total[i]);
    }
    printf("\n");
    for(i = 0; i < 8; i++){
        printf("%e  ", duration[i]);
    }
    printf("\n");

    system("pause");

}

double powNormal(double x, int n) {
    double y = x;
    int i;
    for(i=1; i<n; i++) {
        y *= x;
    }
    return y;
}

//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
double powHardInterative(double x, int n) {
    double y = 1;
    while(n > 0) {
        if(n & 1) {
            y = y * x;
        }
        n /= 2;
        x *= x;
    }
}

double powHardRecursive(double x, int  n) {
    if(n == 0) {
        return 1; //When n = 0
    } else if(n == 1) {
        return x;
    } else {
        if(n & 2) {
            return powHardRecursive(x * x, n/2) * x;
        } else {
            return powHardRecursive(x * x, n/2);
        } //Use the recursive way to realize the algorithm 2
    }
}
```

Declaration

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