浙江大学

Performance Measurement

Fundamentals of Data Structures

Research Project 1

Group ???

??? ??? ???

September 23, 2019

Contents

Chapter 1

Introduction

1.1 Problem Description

Calculate X^N with 2 different algorithms.

1.2 Purpose of Report

In this project, we will first implement the two algorithms calculating X^N. Then we will test their performances and compare them agains each other.

In the end, we will give a proof of their complexities.

Chapter 2

Algorithm Specification

2.1 Algorithm 1

Initialize the variable *product* with X.
Multiply it with X for $N-1$ times.

2.2 Algorithm 2

2.2.1 Recursive

Define $f(X,N)$ as the process for calculating X^N.

$$f(X,N) = \begin{cases} X, & N = 1 \\ f(X^2, N/2), & N > 1, N = 2k \\ X \cdot f(X^2, (N-1)/2), & N > 1, N = 2k+1 \end{cases}$$

Calculate $f(X,N)$ recursively.

2.2.2 Iterative

1. Initialize the variable *product* with 1.
2. If the current N is odd, multiply *product* by X.
3. Divide N by 2, replace the base X with X^2.
4. Repeating process 2 and 3 until N becomes 0.

Chapter 3

Testing Results

All the time is displayed in seconds.

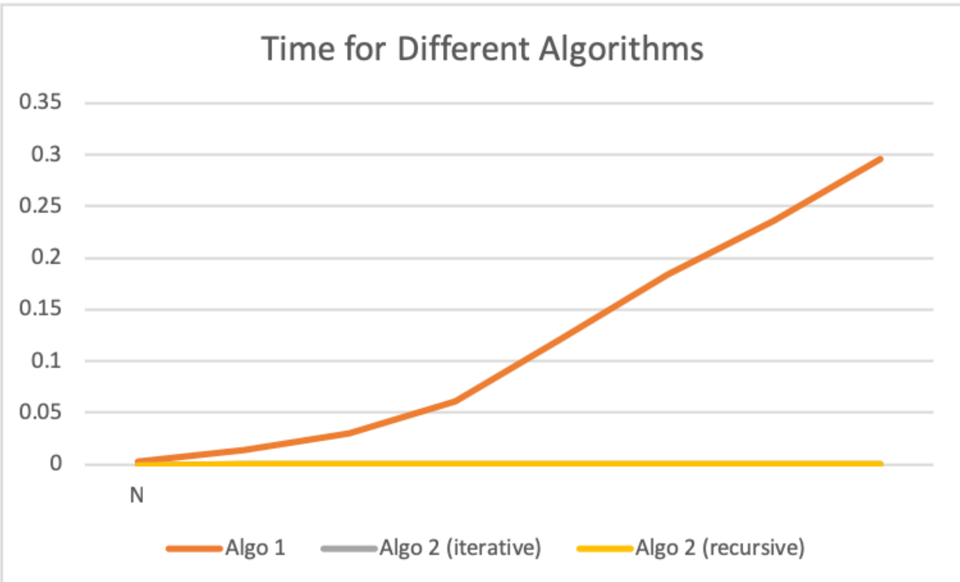
3.1 Time Table

	N	1000	5000	10000	20000
Algo 1	Iterations (K)	10000	10000	10000	10000
	Ticks	30	142	296	609
	Total Time (sec)	0.03	0.142	0.296	0.609
	Duration (sec)	0.003	0.0142	0.0296	0.0609
Algo 2 (iterative)	Iterations (K)	10000000	10000000	10000000	10000000
	Ticks	437	532	547	593
	Total Time (sec)	0.437	0.532	0.547	0.593
	Duration (sec)	0.0000437	0.0000532	0.0000547	0.0000593
Algo 2 (recursive)	Iterations (K)	10000000	10000000	10000000	10000000
	Ticks	1109	1374	1499	1578
	Total Time (sec)	1.109	0.156	0.156	0.172
	Duration (sec)	0.0001109	0.0001374	0.0001499	0.0001578

	N	40000	60000	80000	100000
Algo 1	Iterations (K)	10000	10000	10000	10000
	Ticks	1219	1845	2360	2954
	Total Time (sec)	1.219	1.845	2.36	2.954
	Duration (sec)	0.1219	0.1845	0.236	0.2954
Algo 2 (iterative)	Iterations (K)	10000000	10000000	10000000	10000000
	Ticks	640	625	656	641
	Total Time (sec)	0.64	0.625	0.656	0.641
	Duration (sec)	0.000064	0.0000625	0.0000656	0.0000641
Algo 2 (recursive)	Iterations (K)	10000000	10000000	10000000	10000000
	Ticks	1704	1734	1781	1782
	Total Time (sec)	0.188	0.187	0.188	0.186
	Duration (sec)	0.0001704	0.0001734	0.0001781	0.0001782

3.2 Plots

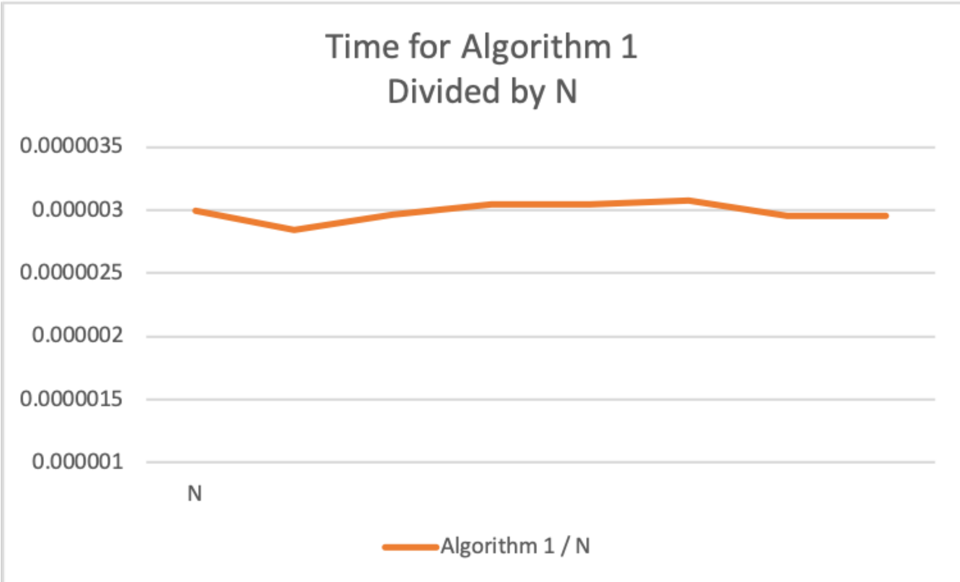
3.2.1 General Comparison



As N becomes larger, the time consumption of Algorithm 1 increases rapidly, and Algorithm 1 becomes much slower than Algorithm 2.

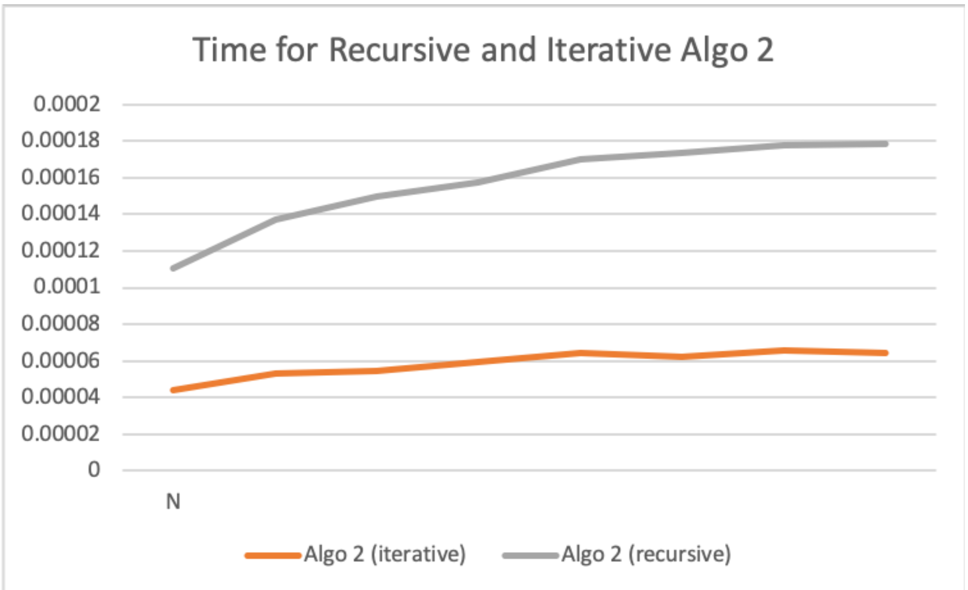
The iterative and recursive versions of Algorithm 2 are too fast compared to Algorithm 1, so they look like the same line in the figure.

3.2.2 Time Complexity of Algorithm 1



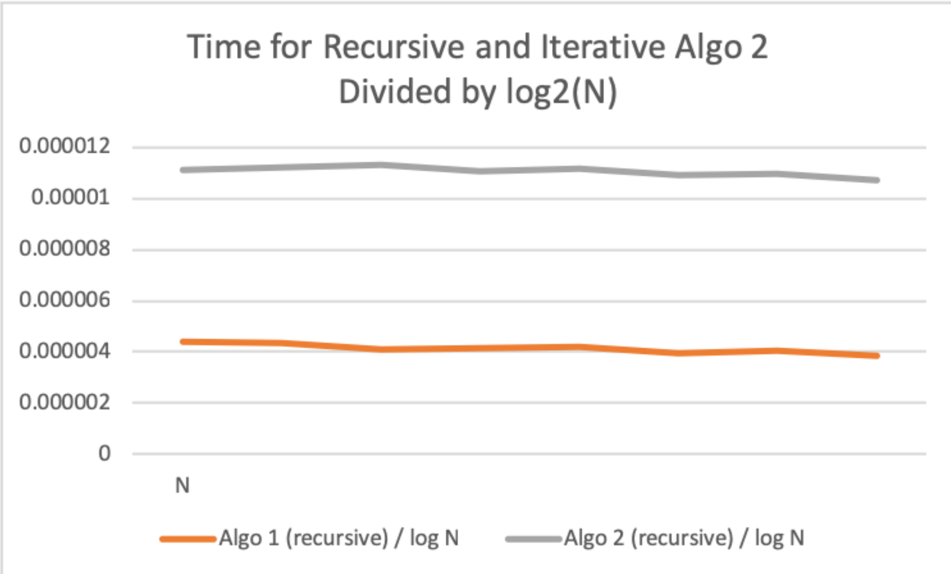
By plotting the time of Algorithm 1 divided by N, the figure looks almost like a straight horizontal line, which indicates that the time complexity of Algorithm 1 is $O(N)$.

3.2.3 Recursive and Iterative Algorithm 2



If we put the running time of two different versions of Algorithm 2 in one figure, it's clear that the iterative version runs faster than the recursive version. However, unlike the situation when we are comparing Algorithm 1 with Algorithm 2, the ratio of their time consumption isn't increasing.

3.2.4 Time Complexity of Algorithm 2



By plotting the time of Algorithm 2 divided by $\log N$, both the iterative and recursive version looks like a horizontal line, which indicates that the time complexity of Algorithm 2 is $O(\log N)$. The different height of the two lines indicates that their time complexities have different constant coefficients.

Chapter 4

Analysis and Comments

4.1 Time Complexity

4.1.1 Algorithm 1

As we did $N-1$ multiplications, the time complexity of Algorithm 1 is $O(N)$.

4.1.2 Algorithm 2

Since every time the problem becomes half as large as the original one, $T(N) = T(N/2) + O(1)$.

We use substitution method to prove its complexity by substituting $T(N/2)$ with $c\log(N/2)$. $T(N) = c\log(N/2) + O(1) = c\log N$. Thus, the time complexity of Algorithm 2 is $O(\log N)$.

4.2 Space Complexity

4.2.1 Algorithm 1

Only a constant number of variables are required no matter how large N is, so the space complexity is $O(1)$.

4.2.2 Algorithm 2 (Recursive)

Every time the process is called, constant number of variables are pushed into the system stack. So the space complexity of recursive Algorithm 2 is the same as its time complexity, $O(\log N)$.

4.2.3 Algorithm 2 (Iterative)

Unlike its recursive version, the iterative Algorithm 2 doesn't use more space as N becomes larger, so its space complexity is $O(1)$.

4.3 Further Improvement

For Algorithm 2, the current base of $O(\log N)$ is 2, maybe we can alter the base to a larger integer to improve the time performance.

Appendices

Appendix A

Source Code (in C)

Listing A.1: Project 1.c

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

const int cnt[] = {1000, 5000, 10000, 20000, 40000, 60000, 80000, 100000};      // cnt 为供测试用的指数
const int cases = 8;      // cnt 数组的大小
const double base = 1.0001;      // 供测试用的底数

double algorithm1(double x, int n);      // 算法1, 暴力计算
double algorithm2_iterative(double x, int n);      // 算法2, 迭代版
double algorithm2_recursive(double x, int n);      // 算法2, 递归版
double get_runtime(double (*f)(double x, int n), double x, int n, int k);      // 计算函数 f 运行 k 次的 tick 数

int main(void) {
    int i, j;
    int k1, k2, k3;
    double ticks;
    for (i = 0; i < cases; i++)
    {
        printf("Please input the number of executions for algorithm 1:");
        scanf("%d", &k1);      // 读入运行次数 k1
        ticks = get_runtime(algorithm1, base, cnt[i], k1);      // 测量时间
```

```
        printf("N:%d Ticks:%.0f Total Time(sec):%6f Duration:%f\n", cnt[i], ticks, ticks / CLK_TCK,
                ticks / k1); // 输出答案
        printf("Please input the number of executions for algorithm 2(iterative):");
        scanf("%d", &k2);                // 读入运行次数 k2
        ticks = get_runtime(algorithm2_iterative, base, cnt[i], k2);               // 测量时间
        printf("N:%d Ticks:%.0f Total Time(sec):%6f Duration:%f\n", cnt[i], ticks, ticks / CLK_TCK,
                ticks / k2); // 输出答案
        printf("Please input the number of executions for algorithm 2(recursive):");
        scanf("%d", &k3);                // 读入运行次数 k2
        ticks = get_runtime(algorithm2_recursive, base, cnt[i], k3);               // 测量时间
        printf("N:%d Ticks:%.0f Total Time(sec):%6f Duration:%f\n", cnt[i], ticks, ticks / CLK_TCK,
                ticks / k3); // 输出答案
    }
}

double algorithm1(double x, int n) {
    int i;
    double res = x;
    for (i = 1; i < n; i++) {
        res *= x;                // 执行 N-1 次乘法
    }
    return x;

}

double algorithm2_iterative(double x, int n) {
    double res = 1;
    for (; n; n /= 2, x *= x)            // 迭代，缩小问题规模
        if (n % 2)res = res * x;                // 若 N 为奇数，则要将 X 乘给答案
    return res;

}

double algorithm2_recursive(double x, int n) {
```

```
    if (n == 1)return x;
        // 递归出口
    if (n % 2 == 0)return algorithm2_recursive(x * x, n / 2);
                                                      // 若 N 为偶数，则直接缩小问题规模
    else return algorithm2_recursive(x * x, n / 2) * x;
                                                              // 若 N 为奇数，则要将 X 乘
        给答案
}

double get_runtime(double (*f)(double x, int n), double x, int
   n, int k) { // 接受函数指针 f，计算其执行 k 次的 tick 数
    clock_t start, stop;
    int i;
    start = clock(); // 记录起始时间
    for (i = 0; i < k; i++) {
        f(x, n);
    }
    stop = clock(); // 记录终止时间
    return stop - start;
}
```

Appendix B

Author List and Declaration

Author List

Code: ???
Test: ???
Report: ???

Declaration

We hereby declare that all the work done in this project titled ”Performance Measurement” is of our independent effort as a group.