# Fundamentals of Data Structures

## Laboratory Projects

# Project 1 Report

# Performance Measurement

**Author： Zhou Zimeng**

**Yu Bingjiao**

**Tang Yuwen**

**Date: 2014-09-27**

# Chapter 1: Introduction

The aim of this project is to measure the running time of a program. The program tested in this project is a simple program, which need to input N integers and output them according to the original order.

Analysis of algorithms, evaluating the relative efficiency of algorithms, is a major topic in computer science. We always use big-O notation to denote the computational complexity of algorithms. And we always use big-O notation to indicate the running time of a program, such as O(N), $O(N^2)$, O(NlogN), $O(2^N)$.

Although we have big-O notation, but it's also important to know the real running time. In this project, we use a O(N) program, to measure the running time with different N.

On the other hand, although the time complexity of both programs are O(N), the running time of those two programs with different algorithm will be different. To research this point, we output the integers with using iterative method and recursive method and measure the running time.

# Chapter 2: Algorithm Specification

## I. Data Structure

We use an array to store integers and an int to store the number of integers.

```
int a[Maxsize];
int n;
```

## II. Algorithm I, use the iterative method to print

***Algorithm Description:***

This is a simple algorithm. Just use a for loop to output every data in the array.

***Function (with pseudo-code):***

```
void Iterative(int n)
```

```
for (i = 0;i < n; i++)
  Output a[i];
```

***Argument:***

`int n`, the size of array.

## III. Algorithm II, use the recursive method to print

***Algorithm Description:***

This also is a simple algorithm. It uses recursion to output data. This algorithm is to equally divide the array into two parts by the integer in the middle position. Then recursively print the first half, followed by printing the integer in the middle, and finally another half.

The function `Recursion(l, r)` can output the integer in array between l and r. The function get integer the middle of interval `[l, r]` and then call the function `Recursion(l, mid)`, output `a[mid]` and call function `Recursion(mid + 1, r)`.

When `l >= r`, end the recursion. (if `l = r`, we have already outputted `a[l]`(or `a[r]`).)

***Function (with pseudo-code):***

`void Recursive(int l, int r)`

```
1: void Recursive(int l, int r){
2:    if (l >= r) return;
3:    mid <- (l + r) / 2;
4:    Recursive(l, mid);
5:    print(a[mid]);
6:    Recursive(mid + 1, r);
7: }
```

***Argument:***

`int l, int r` interval[l, r]

## IV. Algorithm III, Read integers.

***Algorithm Description:***

To test the program and to get the running time with different N, We get the data from file. Firstly, we use function `fopen` to open the file. And then, if open failed, print error message. Finally the program read the integers by function `fscanf` until the end of file.

***Function (with pseudo-code):***

```
Void Getfile()
1:    open the file;
2:    if (open failed) Error;
3:    while (not at the end of the file){
4:      read a integer from file;
5:    }
```

# Chapter 3: Testing Results

## I. Testing Environment

Intel Core i7-5800M (Turbo Boost disabled), Battery-Powered (CPU frequency reduced).

TDM-GCC 4.8 (x86-64), on Windows 8.1 64-bit, Debug mode, No optimize selected.

K: Iterations

## II. N * K = 100000 (constant)

***Table:***

| | *N* | 100 | 500 | 1000 | 2000 | 4000 | 6000 | 8000 | 10000 |
|---|---|---|---|---|---|---|---|---|---|
| Iterative version | Iterations (*K*) | 1000 | 200 | 100 | 50 | 25 | 17 | 13 | 10 |
| | Ticks | 4812 | 4780 | 4904 | 4656 | 4699 | 4931 | 5081 | 4767 |
| | Total Time (sec) | 4.812 | 4.78 | 4.904 | 4.656 | 4.699 | 4.931 | 5.081 | 4.767 |
| | Duration (sec) | 0.004812 | 0.023900 | 0.049040 | 0.093120 | 0.187960 | 0.290059 | 0.390846 | 0.476700 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Recursive version | Iterations (*K*) | 1000 | 200 | 100 | 50 | 25 | 17 | 13 | 10 |
| | Ticks | 4784 | 4762 | 4892 | 4679 | 4623 | 4889 | 5082 | 4760 |
| | Total Time (sec) | 4.784 | 4.762 | 4.892 | 4.679 | 4.623 | 4.889 | 5.082 | 4.76 |
| | Duration (sec) | 0.004784 | 0.023810 | 0.048920 | 0.093580 | 0.184920 | 0.287588 | 0.390923 | 0.476000 |

***A large-scale testing:***

N=10000, K=50

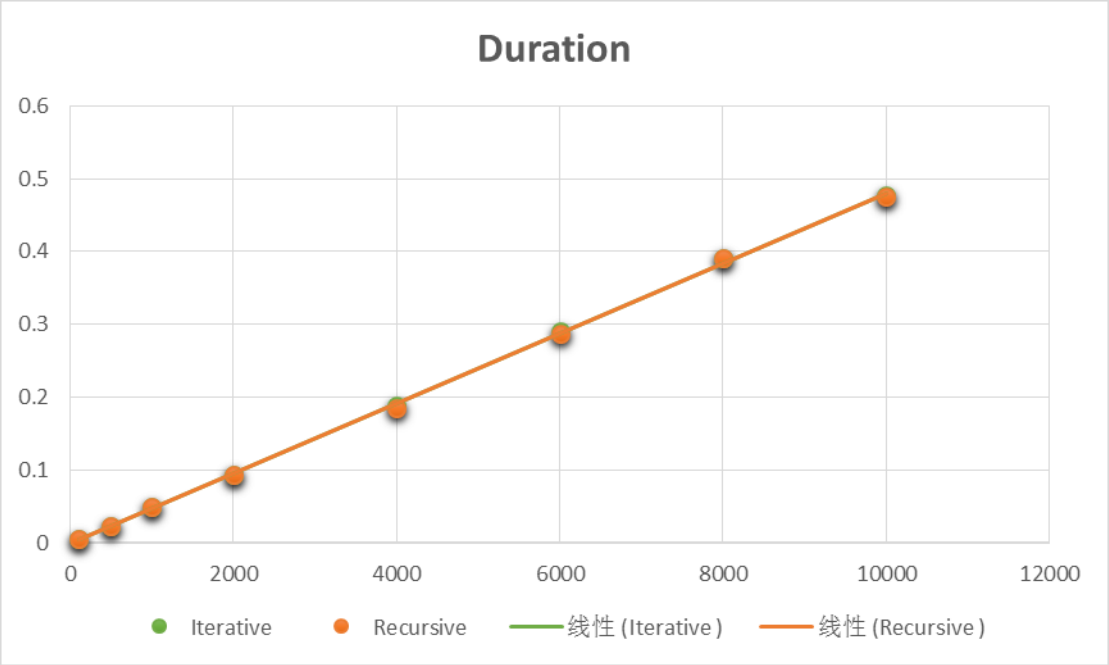
```
Iterative:

Iterations (K) = 50 Ticks (n) = 24456.000000 Total time
(sec) = 24.456000 Duration (sec) = 0.489120

Recursive:

Iterations (K) = 50 Ticks (n) = 24493.000000 Total time
(sec) = 24.493000 Duration (sec) = 0.489860
```

***Graph:***




***Analysis:***

In this test, we controlled the total number of command (n*k=constant). According to data, we can see N, the number of integers, is approximately proportional to the duration.

But there is NO significant difference between the performance of iterative and recursive programs. However, recursive method have a bit advantage on speed (usually in several microseconds). In a large-scale testing, the total time of different method are similar, with iterative has better performance.

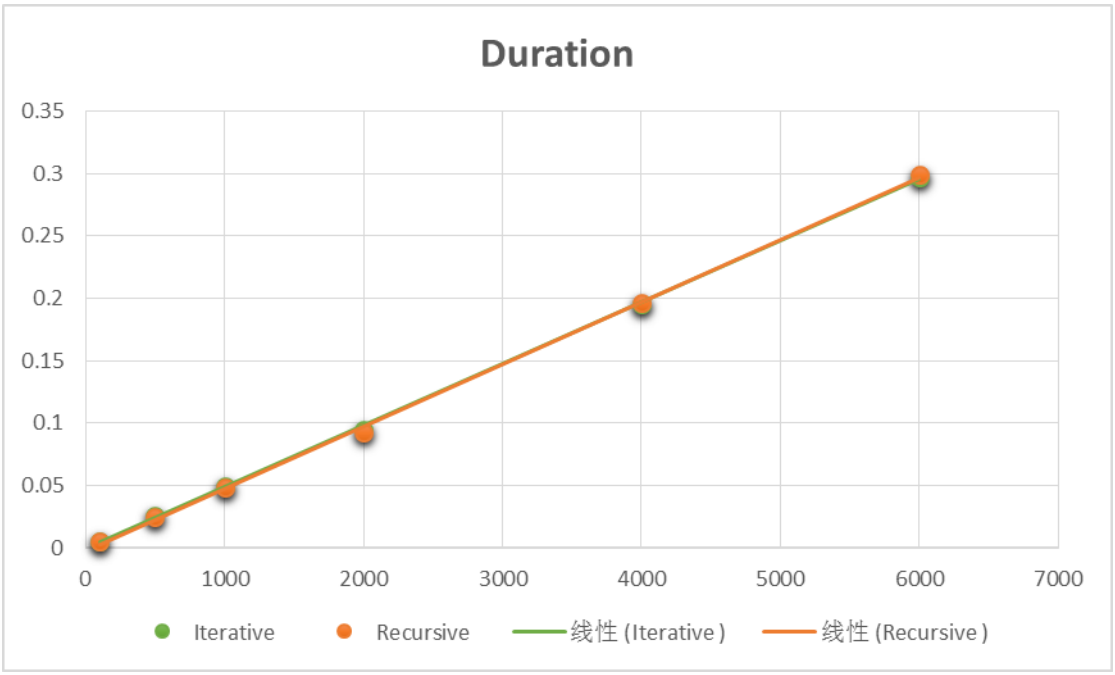
However, this program is quite simple, so the performance bottleneck should not be in programming method (recursive of iterative). The output command, as it should call the console and I/O, should be the performance bottleneck. And as the total number of output command is a constant, both methods cannot optimize the program by reduce the amount of command. So, the performance of recursive and iterative cannot be compared by this program. Also, when optimize is requested, GCC will analysis the code and transfer unnecessary recursive into incursive, which may accelerate the program.

**III.K = 50 (constant)**

***Table:***

| | *N* | 100 | 500 | 1000 | 2000 | 4000 | 6000 | 8000 | 10000 |
|---|---|---|---|---|---|---|---|---|---|
| Iterative version | Iterations (*K*) | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| | Ticks | 281 | 1297 | 2484 | 4731 | 9766 | 14845 | / | / |
| | Total Time (sec) | 0.281 | 1.297 | 2.484 | 4.731 | 9.766 | 14.845 | / | / |
| | Duration (sec) | 0.005620 | 0.025940 | 0.049680 | 0.094620 | 0.195320 | 0.296900 | / | / |
| Recursive version | Iterations (*K*) | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| | Ticks | 250 | 1234 | 2407 | 4643 | 9799 | 14942 | / | / |
| | Total Time (sec) | 0.25 | 1.234 | 2.407 | 4.643 | 9.799 | 14.942 | / | / |
| | Duration (sec) | 0.005000 | 0.024680 | 0.048140 | 0.092860 | 0.195980 | 0.298840 | / | / |

***Graph:***

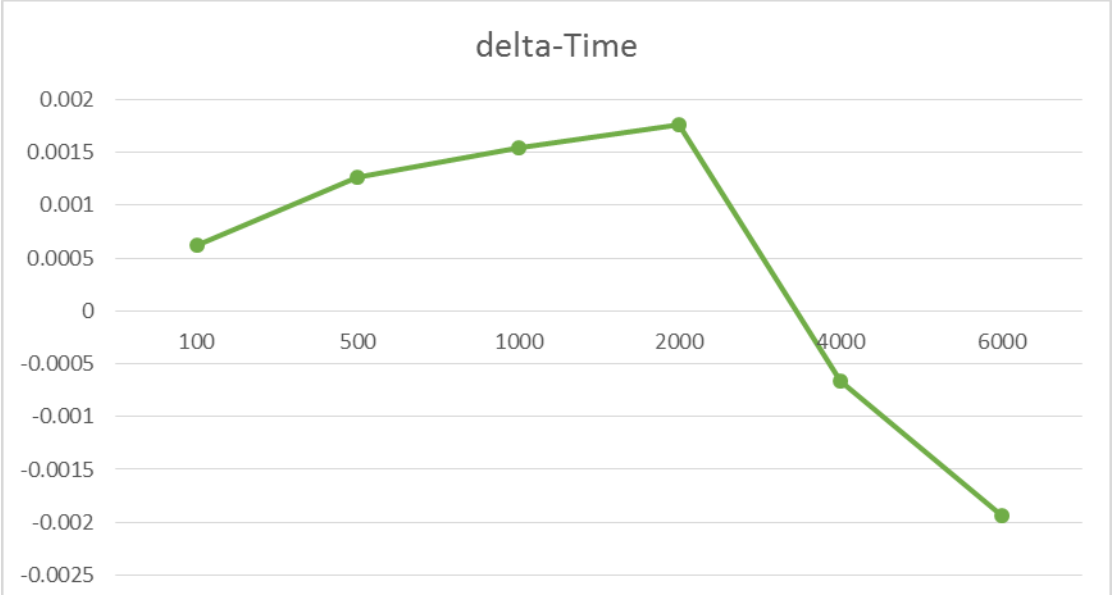


***Analysis:***

Then, we controlled the Iterations (K), and want to analysis whether the number of command will affect the performance. Because we output too much data to the stdout, the duration we measure become inaccurate. We ignored the situation when N=8000 and N = 10000.

As in different input all three arguments are varied, to compare the speed of different methods, we introduce a new variable: the Time difference (Delta-T). By calculating the time difference between iterative and recursive versions, we can compare the performance intuitive.

| N | 100 | 500 | 1000 | 2000 | 4000 | 6000 |
|---|---|---|---|---|---|---|
| Time Difference | 0.00062 | 0.00126 | 0.00154 | 0.00176 | -0.00066 | -0.00194 |




As a result, we can know from the rest data that the time difference is closed to 0, supported our conclusion.

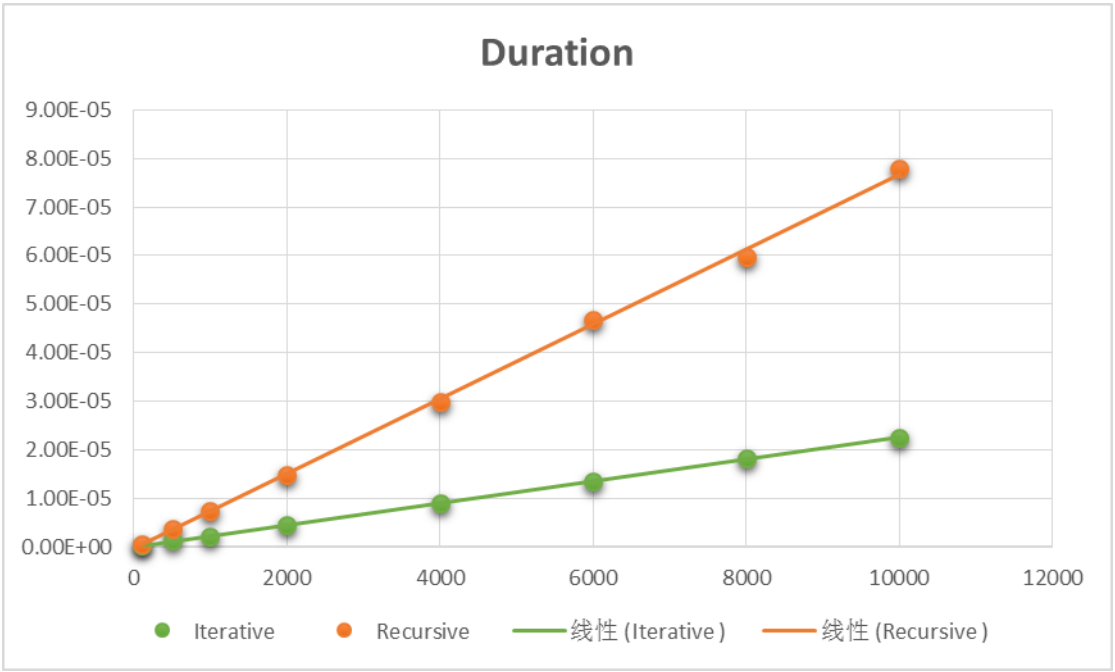
## IV. Without printf

As we all know, the recursive is slower than iterative version. However, we can't get this conclusion from the data we measured before. "Printf" is a complex function. Maybe we can't see "printf" as one unit time. Then, what will happen if we deleted "printf"? We changed the output to a meaningless assignment command.

***Table:***

| | *N* | 100 | 500 | 1000 | 2000 | 4000 | 6000 | 8000 | 10000 |
|---|---|---|---|---|---|---|---|---|---|
| Iterative version | Iterations (*K*) | 50000 | 50000 | 50000 | 50000 | 50000 | 50000 | 50000 | 50000 |
| | Ticks | 16 | 63 | 110 | 225 | 453 | 672 | 906 | 1125 |
| | Total Time (sec) | 0.016 | 0.063 | 0.11 | 0.225 | 0.453 | 0.672 | 0.906 | 1.125 |
| | Duration (sec) | 3.2e-7 | 1.26e-6 | 2.2e-6 | 4.5e-6 | 9.06e-6 | 1.343e-5 | 1.812e-5 | 2.25e-5 |
| Recursive version | Iterations (*K*) | 50000 | 50000 | 50000 | 50000 | 50000 | 50000 | 50000 | 50000 |
| | Ticks | 31 | 187 | 375 | 737 | 1495 | 2344 | 2985 | 3891 |
| | Total Time (sec) | 0.031 | 0.187 | 0.375 | 0.737 | 1.495 | 2.344 | 2.985 | 3.891 |
| | Duration (sec) | 6.2e-7 | 3.74e-6 | 7.5e-6 | 1.474e-5 | 2.99e-5 | 4.686e-5 | 5.97e-5 | 7.782e-5 |

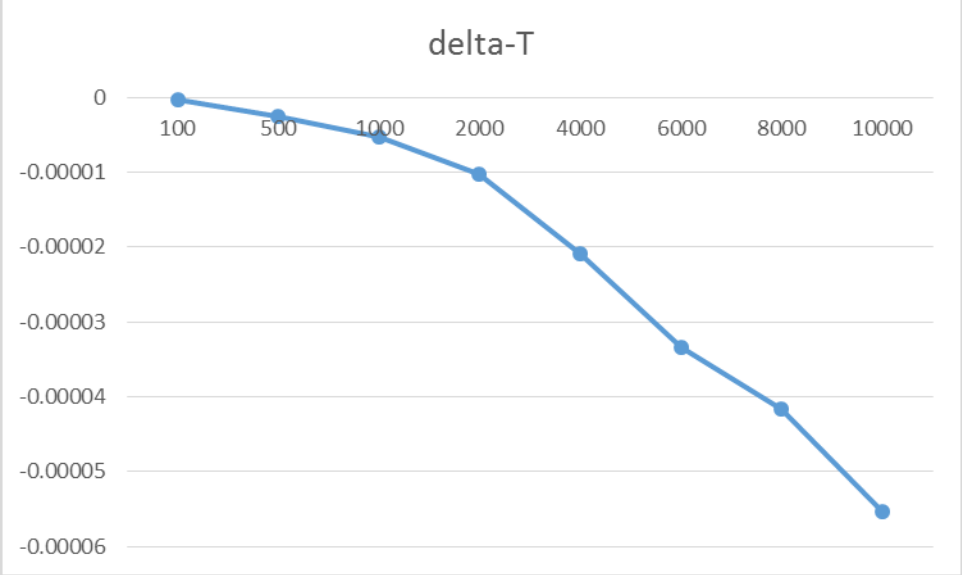
***Graph:***



***Analysis:***

Also, we calculate the delta-T:

| | 100 | 500 | 1000 | 2000 | 4000 | 6000 | 8000 | 10000 |
|---|---|---|---|---|---|---|---|---|
| delta-T | −3E−07 | −2. 5E−06 | −5. 3E−06 | −1E−05 | −2. 1E−05 | −3. 3E−05 | −4. 2E−05 | −5. 5E−05 |




Then, although the running speed is much faster, the result is that iterative version is faster. However, we can see the current delta-T is smaller than that with “printf” by several orders of magnitude, which is so small that reduced accuracy.

If those two function is running without using printf, we can compare the running time with using recursive method or using iterative method.

## V. Further Improvement

The function "printf" is a powerful but complex output method, which brought so many errors into our data. So we need to find a simpler one. We can use "write()" to write a string to stdout directly, without wasting time for format check. Also, writing content to terminal also costs much resource, and the buffer system of terminal interference the testing result. For this problem, we can redirect the stdout stream to a file, skipping the terminal and can get more accurate result.

# Chapter 4: Analysis and Comments

## I. Data Structure

```
clock_t  start1, start2, stop1, stop2;
   //define the beginning and ending time
double   ticks1, ticks2;//define the ticks
char filename[1000];//define the filename
int a[Maxsize]; //define the array printed
int N ,K;//define the integer number N and the iteration number K
```

Those are the most important variables in the program. As we can see, the most large variables is an int array, so the space complexity is *O(N)*.

## II. Algorithm I, use the iterative method to print

```
1: void Iterative(int b)//use the iterative method to print
2: {
3:     int i;
4:     for(i = 0; i < b; i++)
5:         printf("%d\n", a[i]); //print all the integers in order
6:  }
```

Line 3 count for one unit. Line 5 counts for 1 unit per time executed and is executed N times. Line 4 has the hidden costs of initializing *i*, testing $i < b$, and incrementing *i*. The total cost is $2N + 2$. So $T(N) = 3N + 2$, Thus, the time complexity of this function is *O(N)*.

## III. Algorithm II, use the recursive method to print

```
1: void Recursive(int b, int c)//use the recursive method to print
2: {
3:      int mid;
4:      mid = (b + c) / 2; //get the middle integer
5:      if(b < c)
6:       {
7:           Recursive(b, mid);
               //print the array on the left of the middle integer
8:           printf("%d\n", a[mid]);//print the middle integer
9:           Recursive(mid + 1, c);
               //print the array on the right of the middle integer
10:      }
```

```
11:  }
```

Let *T(N)* be the running time for the function `Recursive(0, N)`. If *N = 0* or *N = 1*, then the running time is some constant value. So we can say that *T(0) = T(1) = 1*. When *N > 2*, the time *T(N) = 2T(N/2) + 4*. The work at Line 7 and Line 9 are *T(N/2)*. And 4 accounts for the work at line 4(plus and divide), 5 and 8.

According to Mathematical induction, we get *T(N)* ≈ *N*, so we know the time complex is *O(N)*.

There is a more simple method of estimation of the time complexity. Totally, Line 4 and 5 run *2N-1* times. Line 8 run *N* times. So the this function is *O(N)*.

## IV. Testing program (main program)

```
void Getfile()//read the file and get N and the array
{
    int i=0;
    FILE *fp;
    fp=fopen(filename,"rt");//open the file
    if(fp==NULL) {
        printf("error!\n");
        getch();
        exit(1);
        }
    fscanf(fp,"%d",&N);//read the file and get the integer number N
    while(!feof(fp)) {
        fscanf(fp,"%d",&a[i]);
        i++;
        }//read the file and get the array
    fclose(fp); //close the file
}

void Output(double ticks)//print all the data recorded and calculated
{
    double totalTime, duration;
    totalTime = ticks / CLK_TCK;
    duration = totalTime / K;//caculation

    printf("Iterations (K) = %d Ticks (n) = %f Total time (sec) = %f Duration
(sec) = %f\n", K, ticks, totalTime, duration);//print all the data needed
}
```

```
int main ( )
{
       printf("Please input the file name in the test program: ");
       scanf("%s",filename);//input the name of the file
       printf("Please input the Iteration number K in the test program:
");
       scanf("%d",&K); //input the iteration number K
       Getfile();

       if(N != 0)
       {
           int i;
           start1 = clock();//timing begins
           for(i = 0; i < K; i++)
               Iterative(N);
           stop1 = clock();//timing ends
           ticks1 = stop1-start1;//the ticks recorded

           start2 = clock();
           for(i = 0; i < K; i++)
               Recursive(0, N);
           stop2 = clock();
           ticks2 = stop2-start2;

           printf("Iterative:\n ");
           Output(ticks1);
           printf("Recursive:\n ");
           Output(ticks2); //print all the data recorded and calculated
           return(0);
       }
}
```

Firstly, we ask the user to input the file name and the Iteration number *K*. If the program can't open the file, it will output the error message. Otherwise, use function `Getfile` to read the integers.

And then the program use function `clock()` to get the running time of two print function.

Finally, the program use function `Output` to give the results to users.

## Appendix: Source Code (in C)

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

#define Maxsize 10000
    //define the maximum capacity of the array to be stored and printed

clock_t  start1, start2, stop1, stop2;
    //define the beginning and ending time
double   ticks1, ticks2;//define the ticks
char filename[1000];//define the filename
int a[Maxsize]; //define the array printed
int N ,K;//define the integer number N and the iteration number K

void Iterative(int N);//use the iterative method to print
void Recursive(int b, int c);//use the recursive method to print
void Output(double ticks);//print all the data recorded and calculated
void Getfile(); //read the file and get N and the array

int main ( )
{


       printf("Please input the file name in the test program: ");
       scanf("%s",filename);//input the name of the file
       printf("Please input the Iteration number K in the test program:
");
       scanf("%d",&K); //input the iteration number K
       Getfile();

       if(N != 0)
       {
           int i;
           start1 = clock();//timing begins
           for(i = 0; i < K; i++)
               Iterative(N);
           stop1 = clock();//timing ends
           ticks1 = stop1-start1;//the ticks recorded

           start2 = clock();
           for(i = 0; i < K; i++)
               Recursive(0, N);
```

```
            stop2 = clock();
            ticks2 = stop2-start2;

            printf("Iterative:\n ");
            Output(ticks1);
            printf("Recursive:\n ");
            Output(ticks2); //print all the data recorded and calculated
            return(0);
        }
}


void Iterative(int b)//use the iterative method to print
{
    int i;
    for(i = 0; i < b; i++)
        printf("%d\n", a[i]); //print all the integers in order
}

void Recursive(int b, int c)//use the recursive method to print
{
    int mid;
    mid = (b + c) / 2; //get the middle integer
    if(b < c)
    {
        Recursive(b, mid);
              //print the array on the left of the middle integer
        printf("%d\n", a[mid]);//print the middle integer
        Recursive(mid + 1, c);
              //print the array on the right of the middle integer
    }
}

void Output(double ticks)//print all the data recorded and calculated
{
   double totalTime, duration;
   totalTime = ticks / CLK_TCK;
   duration = totalTime / K;//caculation

   printf("Iterations (K) = %d Ticks (n) = %f Total time (sec) = %f Duration
(sec) = %f\n", K, ticks, totalTime, duration);//print all the data needed
}
```

```
void Getfile()//read the file and get N and the array
{
    int i=0;
    FILE *fp;
    fp=fopen(filename,"rt");//open the file
    if(fp==NULL) {
        printf("error!\n");
        getch();
        exit(1);
        }
    fscanf(fp,"%d",&N);//read the file and get the integer number N
    while(!feof(fp)) {
        fscanf(fp,"%d",&a[i]);
        i++;
        }//read the file and get the array
    fclose(fp); //close the file
}
```

## Declaration

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

## Duty Assignments:

**Programmer:** Tang Yuwen
**Tester:** Yu Bingjiao
**Report Writer:** Zhou Zimeng