Suhyoon Bae

Lab 04

CIS 120 Data Structures

2020-03-03

Lab 04

Recursion: Sum of Digits, Fibonacci Numbers

Page(s) File

1 Code printouts – DigitPlay.java

2 Screenshots of the output demonstrating the test data

3-5 Code printouts - TestFib and Fib classes

6 Screen shots of TestFib

6 Call tree for the recursive Fib(5).

7 Elapsed times of the recursive vs iterative versions of Fib.

**Part 1: Counting and Summing Digits in an Integer**

**1. Printouts of DigitPlay.java**

**import** java.util.Scanner;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// DigitPlay.java

//

// Finds the number of digits in a positive integer.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **class** DigitPlay

{

**public** **static** **void** main (String[] args)

{

Scanner conIn = **new** Scanner(System.***in***);

**int** num; //a number

System.***out***.println ();

System.***out***.print ("Please enter a positive integer: ");

**if** (conIn.hasNextInt())

num = conIn.nextInt();

**else**

{

System.***out***.println("Error: you must enter an integer.");

System.***out***.println("Terminating program.");

**return**;

}

System.***out***.println();

**if** (num <=0)

System.***out***.println("Error: It is not positive integer."

+ " Enter positive integer.");

**else**

{

**if** (*sumDigits*(num)%7==0)

System.***out***.println(num+"---ok");

**else**

System.***out***.println(num+"---error");

}

**if** (num <= 0)

System.***out***.println ( num + " isn't positive -- start over!!");

**else**

{

// Call numDigits to find the number of digits in the number

// Print the number returned from numDigits

System.***out***.println ("\nThe number " + num + " contains "

+ *numDigits*(num) + " digits.");

System.***out***.println ();

System.***out***.println("\nThe sum "+ num + " contains "

+ *sumDigits*(num) + " digits.");

}

}

// -----------------------------------------------------------

// Recursively counts the digits in a positive integer

// -----------------------------------------------------------

**public** **static** **int** numDigits(**int** num)

{

**if** (num < 10)

**return** (1);

**else**

**return** (1 + *numDigits*(num/10));

}

**public** **static** **int** sumDigits(**int** num)

{

**int** sum = 0;

**if** (num<10)

**return** (1);

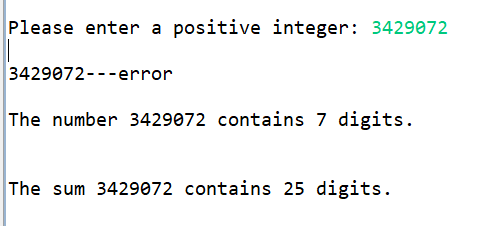
**else**

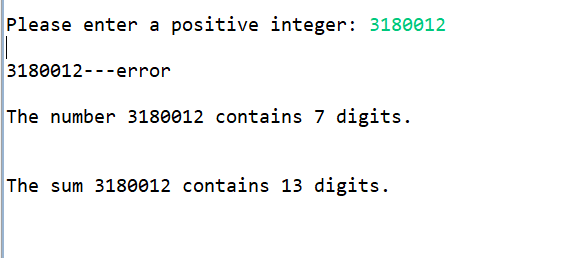
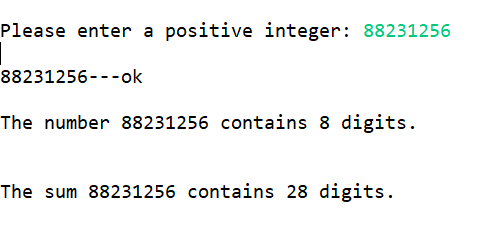
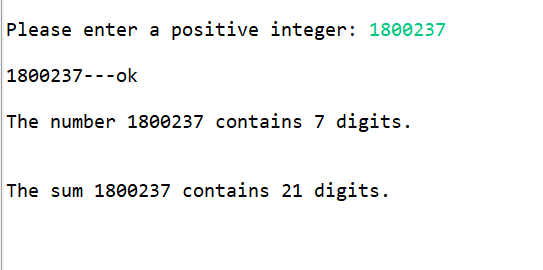
**return** (sum + (num % 10) + *sumDigits*(num/10));

}

}

**2.** **screenshots of the output demonstrating the test data.**





**Part 2: Efficient Computation of Fibonacci Numbers**

**(1)** **Printouts of the TestFib and Fib classes.**

**a. Fib**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Fib.java

//

// A utility class that provide methods to compute elements of the

// Fibonacci sequence.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **class** Fib

{

//--------------------------------------------------------------

// Recursively computes fib(n)

//--------------------------------------------------------------

**public** **static** **int** fib1(**int** n)

{

**if** (n == 0) {

**return** 0;

}

**else** **if** (n == 1) {

**return** 1;

}

**else** **if** (n >1) {

**return** *fib1*(n-1) + *fib1*(n-2);

}

**else** **if** (n < 0) {

System.***out***.println("Please enter a postive number");

}

**return** 1;

//Fill in code -- this should look very much like the

//mathematical specification

}

**public** **static** **int** fib2(**int** n)

{

**int**[] array = **new** **int**[n+1];

**int** i;

**if** (n==0) {

array[0] = 0;

}

**if** (n == 1) {

array[1] = 1;

}

**else** **if** (n > 1) {

array[0] = 0;

array[1] = 1;

**for** (i = 2; i<array.length; i++)

{

array[i] = array[i-1] + array[i-2];

}

}

**else** **if** (n <0 ) {

System.***out***.println("Plese enter a positive number");

}

**return** array[n];

}

}

**b.TestFib**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// TestFib.java

//

// A simple driver that uses the Fib class to compute the

// nth element of the Fibonacci sequence.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**import** java.util.\*;

**import** java.util.Scanner;

**public** **class** TestFib

{

**public** **static** **void** main(String[] args)

{

**int** n, fib;

Scanner conIn = **new** Scanner(System.***in***);

System.***out***.println ();

System.***out***.print ("Please enter a positive integer: ");

Scanner scan = **new** Scanner(System.***in***);

n = scan.nextInt();

fib = Fib.*fib1*(n);

System.***out***.println("In fib1("+ n +") is " + fib);

fib = Fib.*fib2*(n);

System.***out***.println("In fib2(" + n +") is " + fib);

**if** (conIn.hasNextInt())

n = conIn.nextInt();

**else**

{

System.***out***.println("Error: you must enter an integer.");

System.***out***.println("Terminating program.");

**return**;

}

System.***out***.println();

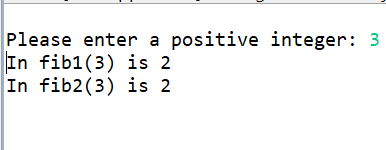
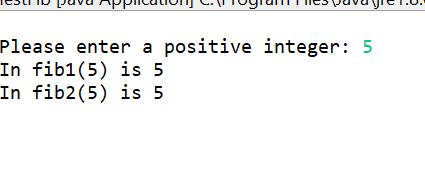
fib = Fib.*fib1*(n);

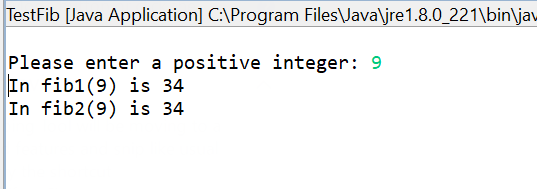
System.***out***.println("Fib(" + n + ") is " + fib);

}

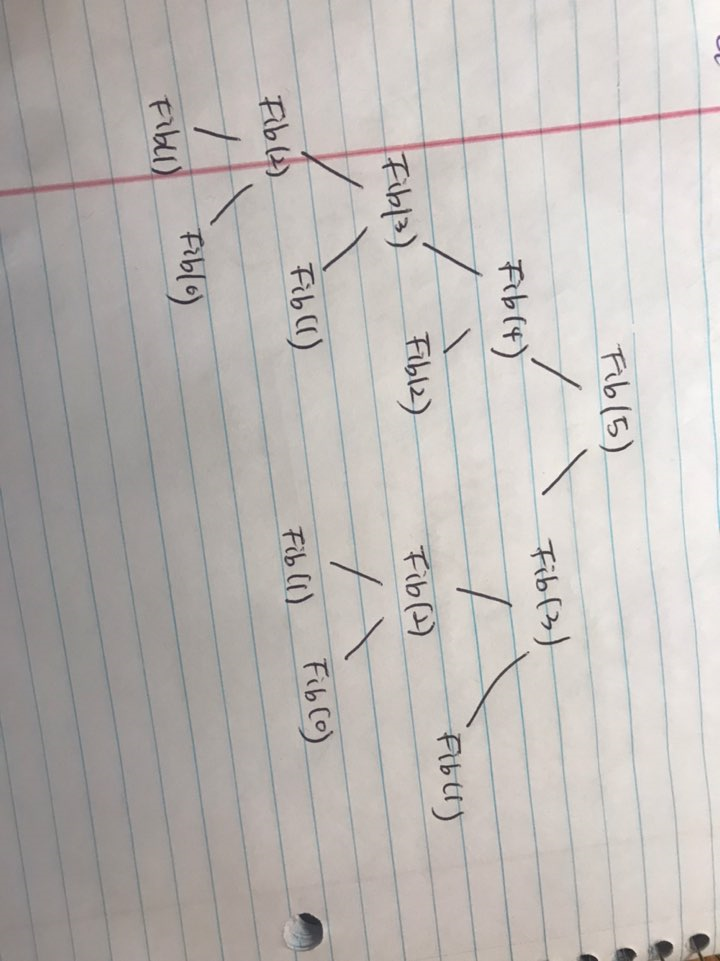
}

**(2) screen shots of the output of TestFib.**



**(3)** **Paper and pencil drawing of the call tree for the recursive Fib(5).**



**(4) A word doc that explains any difference in the** **elapsed times of the recursive vs iterative versions of Fib.**

There are differences in time and space between Recursive and Iterative versions of Fib.

In Recursion method, there will be two recursive calls if number is larger than 1:

**else** **if** (n >1) {

**return** *fib1*(n-1) + *fib1*(n-2);

}

If the numbers are not 0, 1,2 ,3, 4, 5, the outcome will be: 0, 1, 1, 2, 3, 5.

If the numbers are much greater like 100, it will take a longer time to get an outcome compare to when the numbers where small. Therefore, we can say it has a poor performance and it is because of heavy push-pop of the stack memory in each recursive call. The tree for the recursive Fib(5) shows the maximum depth of the recursion tree, which is the required space.

In Iteration method, getting the outcome of 100th Fibonacci number is much faster compare to the Recursion method because array is used to store the Fibonacci number. Therefore, the amount of required space is the same for any Fibonacci numbers, and the time complexity would be O(n).