Andreas Landgrebe

```
import com.clarkware.Profiler;
import java.io.*;

public class UseFibonacci
{

   public static void main(String[] args)
   {

        System.out.println("Begin experiment with different Fibonacci " +
                           "implementations ...");
        System.out.println();

        // extract the value that was passed on the command
        // line; this is the nth fibonacci number that
        // we must calculate in the three different fashions
        Integer Num = new Integer(args[0]);
        int num = Num.intValue();Recursion

        // determine which algorithm we are supposed to benchmark
        String chosen = args[1];
        String type = args[2];
Recursion
        if( chosen.equals("recursive") ||
            chosen.equals("all") )
            {

                // 1. RECURSIVE fibonacci (int)
                if(type.equals("int") || type.equals("all")){

                    Profiler.begin("RecursiveFibonacciInt");
                    int recursiveFib = RRecursionecursiveFibonacci.fib(num);
                    Profiler.end("RecursiveFibonacciInt");

                    System.out.println("(Recursive/int) The " + num +
                                       "th Fibonacci " +
                                       "number = " + recursiveFib + ".");
                }
                // 1. RECURSIVE fibonacci (long)
                if(type.equals("long") || type.equals("all")){

                    Profiler.begin("RecursiveFibonacciLong");
                    long recursiveFibLong = RecursiveFibonacci.fibLong(num);
                    Profiler.end("RecursiveFibonacciLong");

                    System.out.println("(Recursive/long) The " + num +
                                       "th Fibonacci " +
```

```
Recursion                                   "number = " + recursiveFibLong + ".");

                        }
                }
            if( chosen.equals("iterative") ||
                chosen.equals("all") )
                {


                    // 2. ITERATIVE fibonacci (int)
                    if(type.equals("int") || type.Recursionequals("all")){

                        Profiler.begin("IterativeFibonacciInt");
                        int iterativeFib = IterativeFibonacci.fibRecursion(num);
                        Profiler.end("IterativeFibonacciInt");

                        System.out.println("(Iterative/int) The " + num +
                                           "th Fibonacci " +
                                           "number = " + iterativeFib + ".");
                    }
                    // 2. ITERATIVE fibonacci (long)
                    if(type.equals("long") || type.equals("all")){

                        Profiler.begin("IterativeFibonacciLong");
                        long iterativeFibLong = IterativeFibonacci.fibLong(num);
                        Profiler.end("IterativeFibonacciLong");
                    Recursion
                        System.out.println("(Iterative/long) The " + num +
                                           "th Fibonacci " +
                                           "number = " + iterativeFibLong + ".");
                    }
                }

            System.out.println();
            Profiler.print(new PrintWriter(System.out));

            System.out.println("... End experiment with different Fibonacci " +
                               "implementations");

        }

    }
```

Andreas Landgrebe
Lab 2 Report 2/4/2013

#7	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Means
Iterative Int	2.0(ms)	2.0(ms)	1.0(ms)	1.0(ms)	2.0(ms)	1.6(ms)
Iterative Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)
Recursive Int	2.0(ms)	2.0(ms)	2.0(ms)	2.0(ms)	1.0(ms)	1.6(ms)
Recursive Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)

In the testing of the 7th number, I noticed that means for the iterative int and the recursive int were the same. The Iterative long and the Recursive long also had the same mean. These means have the same value due to testing such a low value. When I am testing a bigger value, the means and the trials will vary more.

#8	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Means
Iterative Int	1.0(ms)	1.0(ms)	1.0(ms)	1.0(ms)	1.0(ms)	1.0(ms)
Iterative Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)
Recursive Int	1.0(ms)	1.0(ms)	1.0(ms)	1.0(ms)	1.0(ms)	1.0(ms)
Recursive Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)

In the testing of the 8th number, I noticed that the means for the iterative int and the recursive int had the same value. I also noticed that the iterative long and the recursive long also the same value for the mean. The iterative and the recursive have the same mean at int and long because the numbers that we are testing in these trials are too low for the number to have varied numbers. When I am testing bigger values, the means and the trials will vary more.

#13	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Means
Iterative Int	2.0(ms)	1.0(ms)	1.0(ms)	2.0(ms)	1.0(ms)	1.4(ms)
Iterative Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)
Recursive Int	1.0(ms)	2.0(ms)	2.0(ms)	2.0(ms)	2.0(ms)	1.8(ms)
Recursive Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)

In the testing of the 13th number, I noticed that the iterative long and the recursive long had the same value at the mean. I also noticed that the iterative int and the recursive int did not have the same mean value. This shows that after five trials of testing, the iterative method takes a shorter time than the recursive int.

#15	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Means
Iterative Int	1.0(ms)	2.0(ms)	1.0(ms)	1.0(ms)	1.0(ms)	1.2(ms)
Iterative Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)
Recursive Int	1.0(ms)	1.0(ms)	1.0(ms)	2.0(ms)	1.0(ms)	1.2(ms)
Recursive Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	1.0(ms)	0.2(ms)

In the testing of the 15th number, I noticed that the iterative int and the recursive int have the same mean value. I also noticed that the iterative long have a lower mean value than the recursive mean value. After five trial of testing, I can conclude that the iterative long method takes a shorter period than the recursive long.

#18	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Means
Iterative Int	1.0(ms)	1.0(ms)	2.0(ms)	1.0(ms)	1.0(ms)	1.2(ms)
Iterative Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)
Recursive Int	2.0(ms)	2.0(ms)	1.0(ms)	2.0(ms)	2.0(ms)	1.4(ms)
Recursive Lon	1.0(ms)	1.0(ms)	0.0(ms)	0.0(ms)	1.0(ms)	0.6(ms)

In the testing of the 18th number, I noticed that the value of the mean for the iterative int took a shorter than the value of the mean for the recursive int. I also noticed that the value of the mean for the iterative long was a lower time than the value of the mean for the recursive long. Due to the five trials of testing, I conclude the method of the iterative int and long takes a shorter time than the method of the recursive int and long.

#23	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Means
Iterative Int	1.0(ms)	1.0(ms)	2.0(ms)	1.0(ms)	2.0(ms)	1.4(ms)
Iterative Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)
Recursive Int	7.0(ms)	7.0(ms)	8.0(ms)	9.0(ms)	8.0(ms)	7.8(ms)
Recursive Lon	2.0(ms)	3.0(ms)	3.0(ms)	3.0(ms)	2.0(ms)	2.6(ms)

In the testing of 23, I noticed that the mean for the value of the iterative int had a shorter time than the recursive int mean value. I also noticed that the value of the iterative long value of the mean has a shorter time than the mean value of the recursive long. From the five trials of testing, I conclude that the iterative method takes a shorter time than the recursive method. I can also conclude that the int method takes a longer time than the long method.

#27	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Means
Iterative Int	2.0(ms)	1.0(ms)	2.0(ms)	2.0(ms)	1.0(ms)	1.6(ms)
Iterative Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)
Recursive Int	7.0(ms)	9.0(ms)	10.0(ms)	9.0(ms)	8.0(ms)	8.6(ms)
Recursive Lon	4.0(ms)	3.0(ms)	4.0(ms)	3.0(ms)	3.0(ms)	3.4(ms)

In the testing of 27, I noticed that as the number increase for each of the number that I am testing, the time for the trials and means are only getting bigger. These number are only increasing because the iterative and recursive methods have to take in more numbers in order to get the correct answer in the Fibonacci sequence. I also noticed that after five trials of testing, that the out of the four methods that I am testing, I noticed that the recursive int takes the longest and that the iterative long takes the longest.

#34	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Means
Iterative Int	2.0(ms)	1.0(ms)	1.0(ms)	2.0(ms)	2.0(ms)	1.6(ms)
Iterative Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)
Recursive Int	40.0(ms)	39.0(ms)	41.0(ms)	35.0(ms)	40.0(ms)	41.0(ms)
Recursive Lon	34.0(ms)	35.0(ms)	36.0(ms)	45.0(ms)	35.0(ms)	35.0(ms)

After running the trials of 34, I am seen a pattern that is recovering. I noticed that the iterative method takes a shorter time than the recursive method. I also noticed that the method that takes the shortest time is the iterative long and the one that takes the longest time is the recursive int. I also noticed that int takes longer than the long. The reason for this is because the long takes in the memory to be called in and from this, it will have more memory to process the Fibonacci sequence.

#36	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Means
Iterative Int	2.0(ms)	2.0(ms)	2.0(ms)	1.0(ms)	1.0(ms)	1.6(ms)
Iterative Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)
Recursive Int	90.0(ms)	90.0(ms)	90.0(ms)	93.0(ms)	94.0(ms)	91.4(ms)
Recursive Lon	91.0(ms)	87.0(ms)	90.0(ms)	87.0(ms)	86.0(ms)	88.2(ms)

After running the trials of 36, I noticed the same pattern that is occurring. I notice that the iterative method takes shorter than the recursive. I also see that the method that is taking the shortest is the iterative long and the one that is taking the longest is the recursive int. I also see that the int is taking longer than the int.

#42	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Means
Iterative Int	2.0(ms)	1.0(ms)	2.0(ms)	1.0(ms)	1.0(ms)	1.4(ms)
Iterative Long	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)	0.0(ms)
Recursive Int	1499(ms)	1495.0(ms)	1500.0(ms)	1506.0(ms)	1497.0(ms)	1499.4(ms)
Recursive Long	1509.0(ms)	1504.0(ms)	1506.0(ms)	1509.0(ms)	1508.0(ms)	1507.2(ms)

After running the trials of 42, I notice some results that I have not encountered before. I notice that the iterative takes a shorter time than the recursive method. I also notice that the iterative long is the method that takes the shortest amount of time. The unusual result that I found in the chart was that the recursive long took a longer time than the recursive int.

Script started on Sat 02 Feb 2013 02:38:53 PM EST
#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$ exit####java UseFibonacci 8 all
all\##[K##########[1@15#[C#[C#[C#[C#[C#[C#[C#[C#########3#[C#[C#[C#[C#[C#[C#[C#[C#########[1P7#[C#[C#[C#[C#[C#[C#[C#[C
Begin experiment with different Fibonacci implementations ...

(Recursive/int) The 7th Fibonacci number = 13.
(Recursive/long) The 7th Fibonacci number = 13.
(Iterative/int) The 7th Fibonacci number = 13.
(Iterative/long) The 7th Fibonacci number = 13.

IterativeFibonacciInt:
  count = 1 total = 2 (ms) average = 2.0 (ms)

RecursiveFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

IterativeFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

RecursiveFibonacciInt:
  count = 1 total = 2 (ms) average = 2.0 (ms)

... End experiment with different Fibonacci implementations

#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$ java UseFibonacci 13 all all
Begin experiment with different Fibonacci implementations ...

(Recursive/int) The 13th Fibonacci number = 233.
(Recursive/long) The 13th Fibonacci number = 233.
(Iterative/int) The 13th Fibonacci number = 233.
(Iterative/long) The 13th Fibonacci number = 233.

IterativeFibonacciInt:
  count = 1 total = 1 (ms) average = 1.0 (ms)

RecursiveFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

IterativeFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

RecursiveFibonacciInt:
  count = 1 total = 2 (ms) average = 2.0 (ms)

... End experiment with different Fibonacci implementations

#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$ java UseFibonacci 13 all
all###########[1P7#[C#[C#[C#[C#[C#[C#[C#[C############################exit#[K####java UseFibonacci 8 all
all\##[K##########[1@15#[C#[C#[C#[C#[C#[C#[C#[C#########3#[C#[C#[C#[C#[C#[C#[C#[C#########5#[C#[C#[C#[C#[C#[C#[C#[C
Begin experiment with different Fibonacci implementations ...

(Recursive/int) The 15th Fibonacci number = 610.
(Recursive/long) The 15th Fibonacci number = 610.
(Iterative/int) The 15th Fibonacci number = 610.
(Iterative/long) The 15th Fibonacci number = 610.

IterativeFibonacciInt:
  count = 1 total = 1 (ms) average = 1.0 (ms)

RecursiveFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

IterativeFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

RecursiveFibonacciInt:
  count = 1 total = 1 (ms) average = 1.0 (ms)

... End experiment with different Fibonacci implementations

#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$ java UseFibonacci 8 all all
Begin experiment with different Fibonacci implementations ...

(Recursive/int) The 8th Fibonacci number = 21.
(Recursive/long) The 8th Fibonacci number = 21.
(Iterative/int) The 8th Fibonacci number = 21.
(Iterative/long) The 8th Fibonacci number = 21.

IterativeFibonacciInt:
  count = 1 total = 1 (ms) average = 1.0 (ms)

RecursiveFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

IterativeFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

RecursiveFibonacciInt:
  count = 1 total = 1 (ms) average = 1.0 (ms)

... End experiment with different Fibonacci implementations

```
#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$ java UseFibonacci 8 all all
Begin experiment with different Fibonacci implementations ...

(Recursive/int) The 8th Fibonacci number = 21.
(Recursive/long) The 8th Fibonacci number = 21.
(Iterative/int) The 8th Fibonacci number = 21.
(Iterative/long) The 8th Fibonacci number = 21.

IterativeFibonacciInt:
  count = 1  total = 1 (ms)  average = 1.0 (ms)

RecursiveFibonacciLong:
  count = 1  total = 0 (ms)  average = 0.0 (ms)

IterativeFibonacciLong:
  count = 1  total = 0 (ms)  average = 0.0 (ms)

RecursiveFibonacciInt:
  count = 1  total = 1 (ms)  average = 1.0 (ms)

... End experiment with different Fibonacci implementations
#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$ java UYse##[K##[K##[KseFibonacci 18 all all
Begin experiment with different Fibonacci implementations ...

(Recursive/int) The 18th Fibonacci number = 2584.
(Recursive/long) The 18th Fibonacci number = 2584.
(Iterative/int) The 18th Fibonacci number = 2584.
(Iterative/long) The 18th Fibonacci number = 2584.

IterativeFibonacciInt:
  count = 1  total = 1 (ms)  average = 1.0 (ms)

RecursiveFibonacciLong:
  count = 1  total = 1 (ms)  average = 1.0 (ms)

IterativeFibonacciLong:
  count = 1  total = 0 (ms)  average = 0.0 (ms)

RecursiveFibonacciInt:
  count = 1  total = 2 (ms)  average = 2.0 (ms)

... End experiment with different Fibonacci implementations
```

#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$ Use
##[K##[K##[K##[K#jaba##[K##[K##[Kva##[K##[Kava UseFb##[Kiob##[K##[Kbonacci 23 all all
Begin experiment with different Fibonacci implementations ...

(Recursive/int) The 23th Fibonacci number = 28657.
(Recursive/long) The 23th Fibonacci number = 28657.
(Iterative/int) The 23th Fibonacci number = 28657.
(Iterative/long) The 23th Fibonacci number = 28657.

IterativeFibonacciInt:
  count = 1 total = 1 (ms) average = 1.0 (ms)

RecursiveFibonacciLong:
  count = 1 total = 2 (ms) average = 2.0 (ms)

IterativeFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

RecursiveFibonacciInt:
  count = 1 total = 7 (ms) average = 7.0 (ms)

... End experiment with different Fibonacci implementations

#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$ java
USE##[K##[KSE##[K##[KseFibonacci SE##[K##[K##[K 36 ALL ##[K##[K##[K##[K all
##[K##[K##[K##[K##[K##[K ##[K##[K all all
Begin experiment with different Fibonacci implementations ...

(Recursive/int) The 36th Fibonacci number = 14930352.
(Recursive/long) The 36th Fibonacci number = 14930352.
(Iterative/int) The 36th Fibonacci number = 14930352.
(Iterative/long) The 36th Fibonacci number = 14930352.

IterativeFibonacciInt:
  count = 1 total = 2 (ms) average = 2.0 (ms)

RecursiveFibonacciLong:
  count = 1 total = 91 (ms) average = 91.0 (ms)

IterativeFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

RecursiveFibonacciInt:
  count = 1 total = 90 (ms) average = 90.0 (ms)

... End experiment with different Fibonacci implementations

#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$
Use##[K##[K##[K##java UseFibonacci 42 all all

Begin experiment with different Fibonacci implementations ...

(Recursive/int) The 42th Fibonacci number = 267914296.
(Recursive/long) The 42th Fibonacci number = 267914296.
(Iterative/int) The 42th Fibonacci number = 267914296.
(Iterative/long) The 42th Fibonacci number = 267914296.

IterativeFibonacciInt:
  count = 1 total = 2 (ms) average = 2.0 (ms)

RecursiveFibonacciLong:
  count = 1 total = 1509 (ms) average = 1509.0 (ms)

IterativeFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

RecursiveFibonacciInt:
  count = 1 total = 1499 (ms) average = 1499.0 (ms)

... End experiment with different Fibonacci implementations

#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$ java UseFibonacci 27 all all
Begin experiment with different Fibonacci implementations ...

(Recursive/int) The 27th Fibonacci number = 196418.
(Recursive/long) The 27th Fibonacci number = 196418.
(Iterative/int) The 27th Fibonacci number = 196418.
(Iterative/long) The 27th Fibonacci number = 196418.

IterativeFibonacciInt:
  count = 1 total = 2 (ms) average = 2.0 (ms)

RecursiveFibonacciLong:
  count = 1 total = 4 (ms) average = 4.0 (ms)

IterativeFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

RecursiveFibonacciInt:
  count = 1 total = 7 (ms) average = 7.0 (ms)

... End experiment with different Fibonacci implementations

#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$ java UseFibonacci ##[K##[K 354##[K##[K4 all all
Begin experiment with different Fibonacci implementations ...

(Recursive/int) The 34th Fibonacci number = 5702887.

(Recursive/long) The 34th Fibonacci number = 5702887.
(Iterative/int) The 34th Fibonacci number = 5702887.
(Iterative/long) The 34th Fibonacci number = 5702887.

IterativeFibonacciInt:
  count = 1 total = 2 (ms) average = 2.0 (ms)

RecursiveFibonacciLong:
  count = 1 total = 34 (ms) average = 34.0 (ms)

IterativeFibonacciLong:
  count = 1 total = 0 (ms) average = 0.0 (ms)

RecursiveFibonacciInt:
  count = 1 total = 40 (ms) average = 40.0 (ms)

... End experiment with different Fibonacci implementations

#]0;landgrebea@aldenv162: ~/cs112/lab2#landgrebea@aldenv162:~/cs112/lab2$ exit
exit

Script done on Sat 02 Feb 2013 02:45:23 PM EST