Gavin Dennis
CS3130
Project 2

When evaluating the algorithms with low values for *n* (around 40 and lower), the function durations were more or less the same. The program was easily able to compute the right answer in no more than a millisecond. It's even possible that the efficient algorithm is slightly slower here due to the higher overhead of the BigInteger class compared to standard long integers.

```
Efficient Fibonacci Algorithm Analysis by Gavin Dennis.

Takes command-line arguments for input, then calculates the corresponding fibonacci numbers
using two algorithms:
        * The standard recursive algorithm that is taught in 2000 level CS classes
        * The more efficient dynamic programming approach which removes redundancy when computing

What to input for command-line arguments:
        * arg1: the nth fibonacci number to find for the first algorithm
        * arg2: the nth fibonacci number to find for the second algorithm

Evaluating algorithms: Standard Algorithm
Fibonacci number 20: 6765
Time duration of standard algorithm: 0ms

Evaluating algorithms: Efficient Algorithm
Fibonacci number 20: 6765
Time duration of efficient algorithm: 1ms
```

When evaluating with larger values, the standard recursion algorithm struggled much more. This time it took over 42 seconds to compute the answer. However, it was no challenge for the dynamic programming algorithm which computed the same answer in less than a second.

```
> java App 50 50
Efficient Fibonacci Algorithm Analysis by Gavin Dennis.

Takes command-line arguments for input, then calculates the corresponding fibonacci numbers
using two algorithms:
        * The standard recursive algorithm that is taught in 2000 level CS classes
        * The more efficient dynamic programming approach which removes redundancy when computing

What to input for command-line arguments:
        * arg1: the nth fibonacci number to find for the first algorithm
        * arg2: the nth fibonacci number to find for the second algorithm

Evaluating algorithms: Standard Algorithm
Fibonacci number 50: 12586269025
Time duration of standard algorithm: 42091ms

Evaluating algorithms: Efficient Algorithm
Fibonacci number 50: 12586269025
Time duration of efficient algorithm: 0ms
```

I continued to run tests with higher numbers for the second argument to see just how fast the efficient algorithm really is. I kept the first argument at *50* as it was already struggling to compute in a timely manner. With *5000* as the second argument, the function finished in under 5 milliseconds.

```
> java App 50 5000
Efficient Fibonacci Algorithm Analysis by Gavin Dennis.

Takes command-line arguments for input, then calculates the corresponding fibonacci numbers
using two algorithms:
        * The standard recursive algorithm that is taught in 2000 level CS classes
        * The more efficient dynamic programming approach which removes redundancy when computing

What to input for command-line arguments:
        * arg1: the nth fibonacci number to find for the first algorithm
        * arg2: the nth fibonacci number to find for the second algorithm

Evaluating algorithms: Standard Algorithm
Fibonacci number 50: 12586269025
Time duration of standard algorithm: 43900ms

Evaluating algorithms: Efficient Algorithm
Fibonacci number 5000: 3878968454388325633701916308325905312082127714646245106160597214895550139044037
970108229164622106694792934528588829738134831020089549829403614301569114789383642165639441069102145056
413370655865623825465670071252592990385493381392883637834751890876297071203333705292310769300851809384
980180384781399674888176555465378829164426891298038461377896902150229308247566634622492307188332480328
750391303529033045058427011476352422702109346376991040067141748832984228914912731040543287532980442736
768229772449877498745556919077038806370468327948113589737399931101062193081490185708153978543791953056
510761053075688783766033667355445258844886241619210553457493675897849027988234351023599844663934853256
411952221859563060475364645470760330902420806382584929156452876291575759142343809142302917491088984155
985443248659407979357131684169286803954530954538869811466508206686289742063932343848846524098874239587
801976993820317174208932265468879364002630797780058759129671389634214252579116872755600360311370547754
246046399875880469851784086743828631 25
Time duration of efficient algorithm: 4ms
```

For the final test, I tried setting the second argument to 50,000 which resulted in a stack overflow

```
        * arg1: the nth fibonacci number to find for the first algorithm
        * arg2: the nth fibonacci number to find for the second algorithm

Evaluating algorithms: Standard Algorithm
Fibonacci number 5: 5
Time duration of standard algorithm: 0ms

Evaluating algorithms: Efficient Algorithm
Exception in thread "main" java.lang.StackOverflowError
        at App.computeEfficientAlgorithm(App.java:92)
        at App.computeEfficientAlgorithm(App.java:97)
        at App.computeEfficientAlgorithm(App.java:97)
        at App.computeEfficientAlgorithm(App.java:97)
        at App.computeEfficientAlgorithm(App.java:97)
        at App.computeEfficientAlgorithm(App.java:97)
        at App.computeEfficientAlgorithm(App.java:97)
        at App.computeEfficientAlgorithm(App.java:97)
        at App.computeEfficientAlgorithm(App.java:97)
        at App.computeEfficientAlgorithm(App.java:97)
        at App.computeEfficientAlgorithm(App.java:97)
        at App.computeEfficientAlgorithm(App.java:97)
```