# The Billionth Fibonacci Number

## The Billionth Fibonacci Number

Kazi Abid Azad

#### Contents

The Billionth Fibonacci Number	ii
Acknowledgements	i٧
Introduction	٧
Behold the 1,000,000,000 <sup>th</sup> Fibonacci Number	٧i
1,000,000,000 <sup>th</sup> Fibonacci Number	1
Source Code	i
References:	iii

### **Acknowledgements**

The author wish to acknowledge The Project Nayuki.

The "Fast Fibonacci" algorithm that has been used to generate this book has been taken from their website.

Readers can visit

https://www.nayuki.io/page/fast-fibonaccialgorithms

for more details.

I also thank the "Fibonacci Quarterly" – Official Publication of The Fibonacci Association for the invaluable information that they have given for free. Visit:

https://www.fg.math.ca

#### Introduction

The Fibonacci series is the sequence of numbers (also called Fibonacci numbers), where every number is the sum of the preceding two numbers, such that the first two terms are '0' and '1'. In some older versions of the series, the term '0' might be omitted. A Fibonacci series can thus be given as:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . . It can thus be observed that every term can be calculated by adding the two terms before it.

Given the first term, F0 and second term, F1 as '0' and '1' respectively, the third term here can be given as, F2 = 0 + 1 = 1

Similarly,

F3 = 1 + 1 = 2

F4 = 2 + 1 = 3

F5 = 2 + 3 = 5F6 = 3 + 5 = 8

F6 = 3 + 5 = 8 F7 = 5 + 8 = 13

and so on

Therefore, to represent any (n+1)th term in this series, we can give the expression as,  $F_n = F_n-1 + F_n-2$ .

Behold the 1,000,000,000<sup>th</sup> Fibonacci Number

#### Source Code

```
import java.io.File;
import java.io.IOException;
import java.io.lutzeption;
import java.io.OutrputStream;
import java.math.BigInteger;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.file.StandardOpenOption;
import java.util.concurrent.atomic.AtomicBoolean;
 * @author Kazi Abid
public final class Fibonacci {
    private final int
    private BigInteger r
                                       <u>N</u>;
                                       stringReasult;
                                       calculated = new AtomicBoolean(false);
     // Requirement: CUTOFF ≥ 64, or else there will be infinite recursion.
     private static final int CUTOFF
       * @param n
       * athrows IllegalArgumentException
     public Fibonacci(int n)
           if (n < 0)
                throw new IllegalArgumentException("Value should be positive integer only!");
     private void checkAndCalculate() {
    if (!calculated.get()) {
                fastFibonacciDoubling();
                calculated.set(true);
       * https://www.nayuki.io/page/fast-fibonacci-algorithms <br>
     private void fastFibonacciDoubling() {
          BigInteger a = BigInteger.ZERO;
           BigInteger b = BigInteger. ONE;
           int \underline{m} = 0;
for (int bit = Integer.highestOneBit(N); bit \neq 0; bit >>>= 1) {
                BigInteger d = multiply(a, b.shiftLeft(1).subtract(a));
                BigInteger e = multiply(a, a).add(multiply(b, b));
                a = d;
                m *= 2;
// Advance by one conditionally
if ((N & bit) ≠ 0) {
                     BigInteger c = a.add(b);
                      a = b;
                      b = c;
                      m++;
           this.result = a;
           this.stringReasult = a.toString();
```

```
private BigInteger multiply(BigInteger x, BigInteger y) {
     return multiplyKaratsuba(x, y);
 * same result as {@code x.multiply(y)} but should be faster.
 * Oparam x a multiplicand
 * aparam y a multiplicand

* areturn {acode x} times {acode} y

* athrows NullPointerException if {acode x} or {acode y} is {acode null}
private BigInteger multiplyKaratsuba(BigInteger x, BigInteger y) {
  if (x.bitLength() ≤ CUTOFF || y.bitLength() ≤ CUTOFF) { // Base case
    return x.multiply(y);
     } else {
           int n = Math.max(x.bitLength()), y.bitLength());
int half = (n + 32) / 64 * 32; // Number of bits to use for the low part
          BigInteger mask = BigInteger. ONE.shiftLeft(half).subtract(BigInteger.ONE);
BigInteger xlow = x.and(mask);
BigInteger ylow = y.and(mask);
BigInteger xhigh = x.shiftRight(half);
BigInteger yhigh = y.shiftRight(half);
           BigInteger a = multiply(xhigh, yhigh);
          BigInteger b = multiply(xlow.add(xhigh), ylow.add(yhigh));
BigInteger c = multiply(xlow, ylow);
           BigInteger d = b.subtract(a).subtract(c);
return a.shiftLeft(half).add(d).shiftLeft(half).add(c);
public BigInteger val() {
     checkAndCalculate();
public String valString() {
     checkAndCalculate();
      return this.stringReasult;
∂Override
public String toString() {
    checkAndCalculate();
     return valString();
 * Writes - overwrites if present
 * aparam file
 * athrows IOException
public void write(File file) throws IOException {
    checkAndCalculate();
     if (!file.exists())
           file.createM
                            lewFile();
     ByteBuffer buffer = ByteBuffer. allocate(stringReasult.length());
           buffer.put(stringReasult.getBytes());
buffer.flip();
channel.write(buffer);
     }
public static void main(String[] args) throws IOException {
```

#### References:

Kazi Abid Azad | Github

LinkedIn - Kazi Abid Azad

Fibonacci Sequence - Wikipedia

Fast Fibonacci Algorithm - Nayuki

Karatsuma Multiplication - Nayuki

The Fibonacci Quarterly

Fibonacci Calculator

All the knowledge in this World is free. Use the internet wisely!