Why is it a bad idea to use recursion method to find the fibonacci of a number?

A brief introduction to the Fibonacci sequence.
The Fibonacci sequence the sequence of numbers where the first two numbers are 0 and 1, with each subsequent number being defined as the sum of the previous two numbers in the sequence. The Fibonacci sequence looks like this
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 and so on.
The pattern involves totalling the two previous numbers in this fashion: 0 + 1 = 1, 1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5 etc 0 + 1 = 1, 1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5

Finding the Fibonacci of a number is a task that can be done using different solutions one of which is the Recursive solution and the other the Iterative solution.

The Recursive solution requires using recursion ie using a function or method that can call itself. A rercusive function requires a base case, which is the condition in which no recursive call is made ie the exit condition . Although it is a more elegant way of writing the function. Below is a function performing this task using recursion

```
var fibonacci = function(number) {
if (number <= 1) return number; //base case
return Fibonacci(number - 2) + Fibonacci(number – 1);
//recursive call
}
```

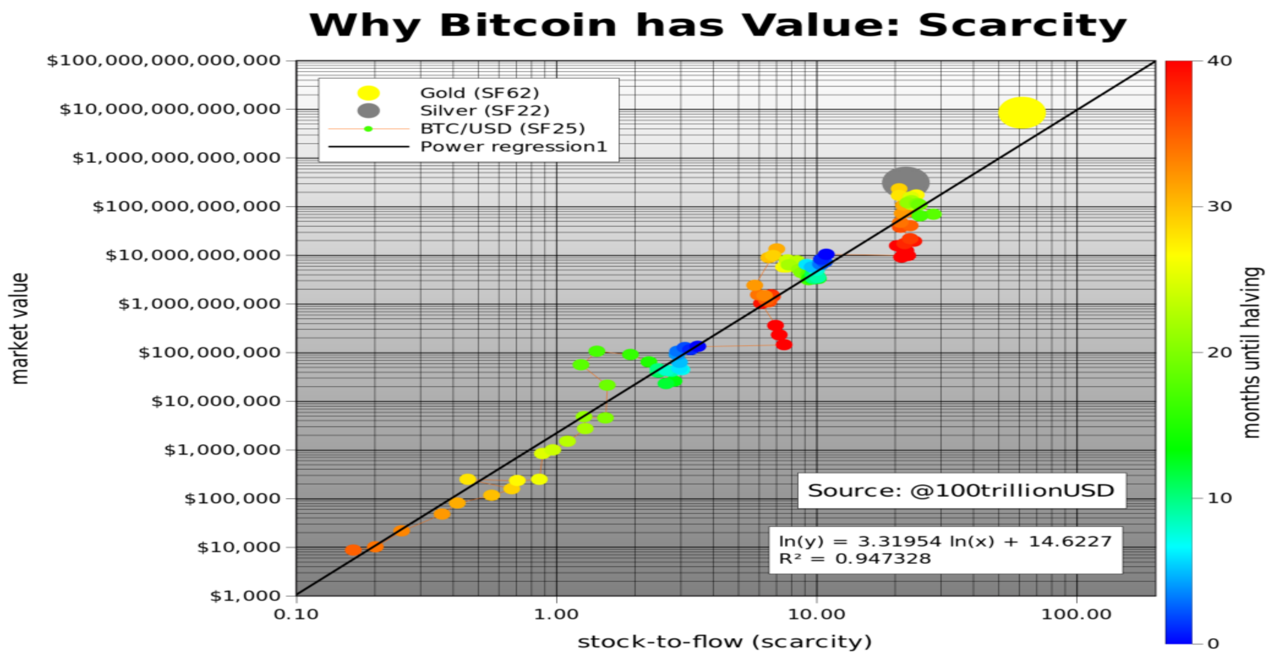
Using recursion however has its drawbacks and is not considered to be the best solution for the task.
Firstly, finding the fibonacci of a large number (say 100th term of the sequence), the function will call itself numerous times (until it hits the base case) which would be detrimental to the performance of the algorithm. These frequent recursive calls means that new execution contexts being added to the call stack, uses more computing power and in turn cause a stack overflow.
Also, a recursive function or algorithm has an exponential time complexity ie its growth doubles with the addition of input data. This is because with every additional element, there is an increase in function calls.

Bitcoin stock-to-flow model and why it is a bad model?

The Stock to Flow model is a pricing model presented PlanB@100trillionUSD who is a Dutch institutional investor with a legal and quantitative finance background. The Stock to Flow (S2F) model which can be applied to natural resources like gold and silver uses scarcity as a means to determine the value of a certain commodity. As bitcoin is seen to be a scarce commodity with just 21 million bitcoins in existence, the S2f model at first glance is well suited to explain bitcoin's value.

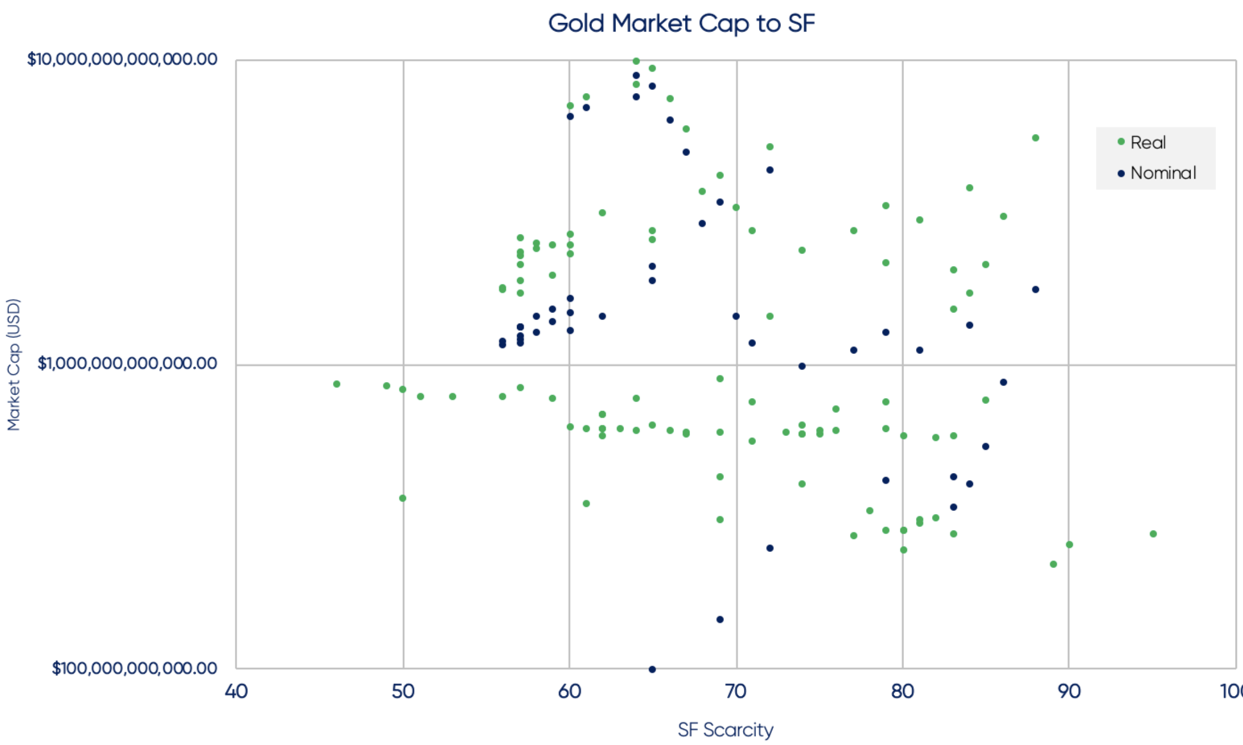
The S2F model/ratio takes into account two things, stock and flow. Stock refers to the current stockpile of the commodity i.e. the amount that has been made and is in circulation or storage. Flow then refers to the amount produced, mined or created annually. Therefore, the SF ratio is the stock/flow of a commodity. Commodities like gold and silver with SF ratios of 62 and 22 respectively has high SF ratio. They both have very high stockpiles and low (maintained) annual production that can't match up the current stockpiles in years. Bitcoin currently has a stock of 17.5 million coins and a flow of 0.7m/yr giving it an SF value of 25. The high SF values of these items give them monetary value. It has a fixed supply and every four years, the bitcoin halvings takes place which reduce supply thereby doubling the SF value.

To argue the case for the S2F model (which is that scarcity and the SF value directly drives the value for bitcoin) PlanB plots SF value of bitcoin, along with gold and silver with their USD market capitalization. He then runs a linear regression using the natural logarithm of Bitcoin's SF metric as the independent variable and the USD market capitalization as the dependent variable.



PlanB then concludes that there is a statistically significant relationship between the USD market capitalization and SF values exists with the likelihood that the relationship between USD market capitalization and SF is caused by chance close to zero, with the two points for gold and silver (which are totally different markets) being in line with the bitcoin values for SF.

However, the model has been shown to be flawed. One of the criticism of the model is the strong assertion it makes that USD market capitalization of a monetary good (in this case gold and silver) is derived directly from its rate of new supply. In this model, gold and silver are depicted at a single data point and is used without any evidence or prior research of both gold and silver. Also, as bitcoins SF ratio increases over 10 years, gold and silver are fixed which shouldn't be the case. Golds SF ratio has been fluctuating for the past 115 years which shows that SF has no direct relationship with golds value.



According to Nico Cordeiro, chief investment officer at Strix Leviathian, PlanB's use of the linear regression to chart the SF model poses a high probability of a spurious result. He states that a 'good' statistical result such as a high R-square doesn't constitute a meaningful finding. This is because in this situation, there is a large degree of freedom in which random data can be fit into a specific outcome.

(Please show your workings). Yara Inc is listed on the NYSE with a stock price of $40 - the company is not known to pay dividends. We need to price a call option with a strike of $45 maturing in 4 months. The continuously-compounded risk-free rate is 3%/year, the mean return on the stock is 7%/year, and the standard deviation of the stock return is 40%/year. What is the Black-Scholes call price?

Using the following

$$V_c = P_0 N_{d_1} - \frac{X}{e^{k_{RF} t}} N_{d_2}$$

and

V_c	= Value of the call option
P_0	= Current Stock Price
k_{RF}	= risk-free rate of interest
t	= time remaining to maturity (fraction of a year)
N_{d1}	= Cumulative area under the normal distribution curve to d_1
N_{d2}	= Cumulative area under the normal distribution curve to d_2
X	= Strike (exercise) price of the option
σ	= volatility (standard deviation) of exchange rate

The value for a put option, can then be found from the following put-call parity relationship.

$$V_p = V_c + \frac{X}{e^{k_{RF} T}} - P_0$$

The value of a call option can be found as follows:

$$d_1 = \frac{\left[\ln\left(\frac{P_0}{X}\right) + (k_{RF} + .5\sigma^2)t\right]}{\sigma\sqrt{t}}$$

where

$$d_2 = d_1 - \sigma\sqrt{t}$$

P_0 = \$40
k_{RF} = 3% = 0.03
t = 4 months = 4/12 = 0.33
X = \$45
σ = 40% = 0.4

Start off by finding the values of d_1 and d_2

$d_1 = (\ln 40/45) + [0.03 + 0.5(0.4)^2] (0.33) / 0.4 (\sqrt{0.33})$
$d_1 = -0.11778 + [0.03 + 0.08](0.33) / 0.4(0.57446)$
$d_1 = -0.11778 + (0.11)(0.33) / 0.22978$
$d_1 = -0.11778 + 0.0363 / 0.22978$
$d_1 = -0.08148 / 0.22798$
$d_1 = -0.3574$

$d_2 = -0.3574 – 0.4 (\sqrt{0.33})$
$d_2 = -0.3574 – 0.22978$
$d_2 = -0.58718$

Using the Standard Normal Distribution table to get the values of Nd_1 and Nd_2

$d_1 \approx -0.36$ and $d_2 \approx -0.59$ making $Nd_1 = 0.3594$ and $Nd_2 = 0.2776$

$V_c = (40)(0.3594) – [(45) / (e^{(0.03 * 0.33)})] (0.2776)$
$V_c = 14.376 – (45/(e^{(0..0099)})) (0.2776)$
$V_c = 14.376 – (45/1.00995)(0.2776)$
$V_c = 14.376 – (44.55666)(0.2776)$
$V_c = 14.376 – 12.3689$
$V_c = \$2.0071$

Over all real numbers, find the minimum value of a positive real number, y such that

y = sqrt($(x+6)^2$ + 25) + sqrt($(x-6)^2$ + 121)

sqrt((x+6)(x+6)+25) + sqrt((x-6)(x-6)+121)

sqrt((x^2+12x+36)+25) + sqrt((x^2-12x+36)+121)

Using b±√b2-4ac / 2a to solve for x

x = 12±√12^2 – 4(36) / 2	x = -12±√(-12^2) – 4(36) / 2
x = 12±√144 – 144 / 2	x = -12±√144 – 144 / 2
x = 12/2	x = -12/2
x = 6	x = -6

sqrt((6^2+12(6)+36)+25) + sqrt(($(-6)^2$-12(-6)+36)+121)

sqrt((36+72+36)+25) + sqrt((36+72+36)+121)

sqrt((144)+25) + sqrt((144)+121)

sqrt(3600) + sqrt(17424)

60 + 132

y = 192

Write a function that takes in a Proth Number and uses Proth's theorem to determine if said number is prime?

Written in JavaScript. Run [here]()

```
function isPower(n){
return (n && !(n & (n-1)))
}

function isProthNumber(n){
var k = 1;
while(k < (n/k)){
if(n % k === 0){
if(isPower(n/k))
return true
}
k = k + 2
}
return false
}

function isPrime(num){
if (num <2){
return false
}
for(let i=2; i<num; i++){
if (num % i === 0){
console.log('No its not prime')
return false
}
}
console.log('yes its prime')
return true
}

const n = 7
function run(){
if (isProthNumber(n - 1)) {
console.log('Yes is a proth number')
isPrime(n)
}
else
console.log('No, its not a proth number')
}


run()
```

```
function isPower(n){
return (n && !(n & (n-1)))
}

function isProthNumber(n){
var k = 1;
while(k < (n/k)){
if(n % k === 0){
if(isPower(n/k))
return true
}
k = k + 2
}
return false
}

function isPrime(num){
if (num <2){
return false
}
for(let i=2; i<num; i++){
if (num % i === 0){
console.log('No its not prime')
return false
}
}
console.log('yes its prime')
return true
}

const n = 7
function run(){
if (isProthNumber(n - 1)) {
console.log('Yes is a proth number')
isPrime(n)
}
else
console.log('No, its not a proth number')
}

run()
```