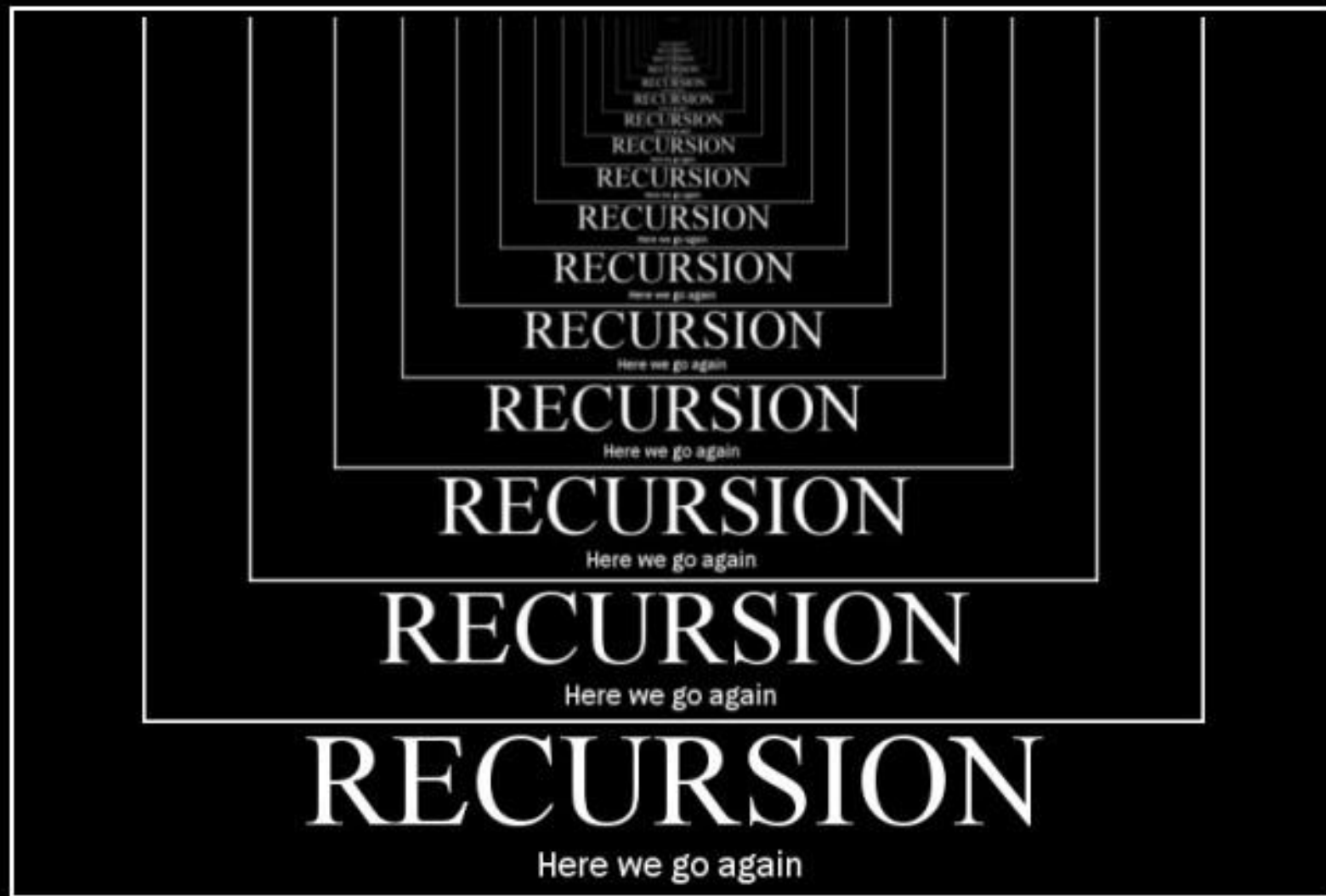


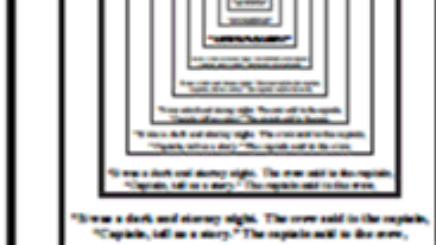
Recursion Introduction



RECURSION

Here we go again





"It was a dark and stormy night. The crew said to the captain,
"Captain, tell us a story." The captain said to the crew,

"It was a dark and stormy night. The crew said to the captain,
"Captain, tell us a story." The captain said to the crew,

"It was a dark and stormy night. The crew said to the captain,
"Captain, tell us a story." The captain said to the crew,

"It was a dark and stormy night. The crew said to the captain,
"Captain, tell us a story." The captain said to the crew,

"It was a dark and stormy night. The crew said to the captain,
"Captain, tell us a story." The captain said to the crew,

"It was a dark and stormy night. The crew said to the captain,
"Captain, tell us a story." The captain said to the crew,

"It was a dark and stormy night. The crew said to the captain,
"Captain, tell us a story." The captain said to the crew,

"It was a dark and stormy night. The crew said to the captain,
"Captain, tell us a story." The captain said to the crew,






Recursion is like a loop.

It repeats a function over and over, progressing to an end.





The picture
is repeated.

It progresses
to the middle.

It gets smaller
and smaller.

“The End” is
the middle; it
is too small to
draw again.

A brief journey into Factorials

Ahhh... Data Management

Suppose that we have 1 thing. How many ways can it be arranged?

Suppose that we have 1 thing. How many ways can it be arranged?

A

Suppose that we have 2 things. How many ways can they be arranged?

Suppose that we have 2 things. How many ways can they be arranged?

AB

BA

Suppose that we have 2 things. How many ways can they be arranged?

AB

BA

$$\boxed{2} \times \boxed{1}$$

Suppose that we have 3 things. How many ways can they be arranged?

Suppose that we have 3 things. How many ways can they be arranged?

ABC ACB

BAC BCA

CBA CAB

Suppose that we have 3 things. How many ways can they be arranged?

ABC

ACB

BAC

BCA

CBA

CAB

$$3 \times 2 \times 1$$

Suppose that we have 4 things. How many ways can they be arranged?

Suppose that we have 4 things. How many ways can they be arranged?

ABCD ABDC ACBD ACDB ADCB ADBC

BACD BADC BCAD BCDA BDCA BDAC

CABD CBAD CADB CABD CDAB CDBA

DABC DBAC DBAC DBCA DCAB DCBA

Suppose that we have 4 things. How many ways can they be arranged?

ABCD ABDC ACBD ACDB ADCB ADBC
BACD BADC BCAD BCDA BDCA BDAC
CABD CBAD CADB CABD CDAB CDBA
DABC DBAC DBAC DBCA DCAB DCBA

$$4 \times 3 \times 2 \times 1$$

A **factorial** is the product of an integer and all the integers below it.

$$1! = 1 = 1$$

$$2! = 2 = 1 \times 2$$

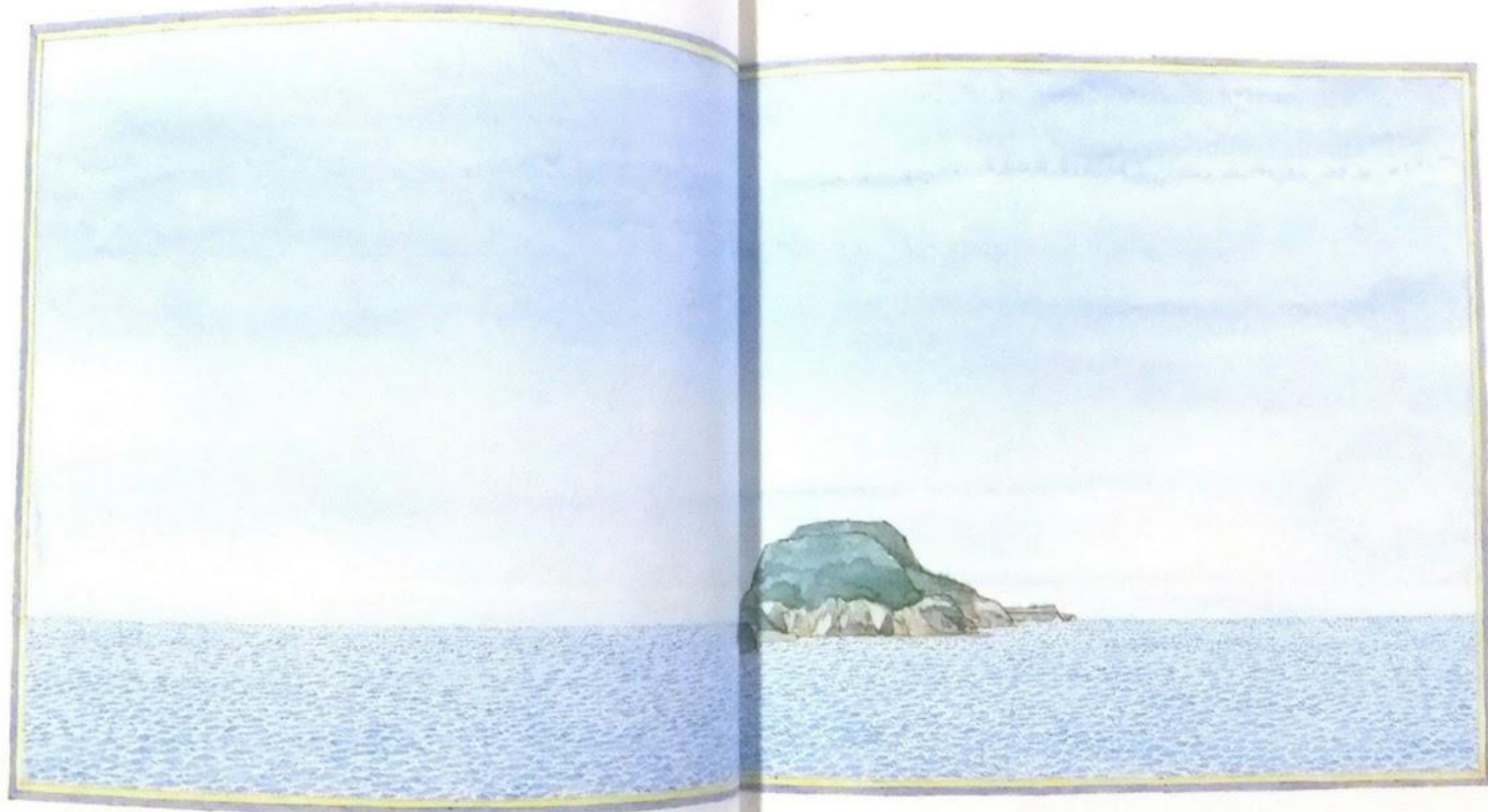
$$3! = 6 = 1 \times 2 \times 3$$

$$4! = 24 = 1 \times 2 \times 3 \times 4$$

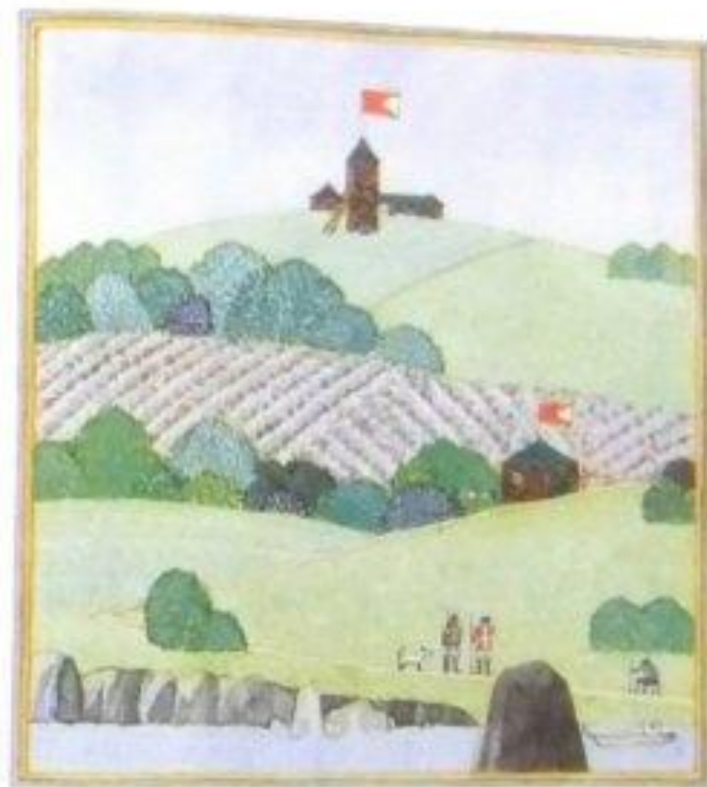
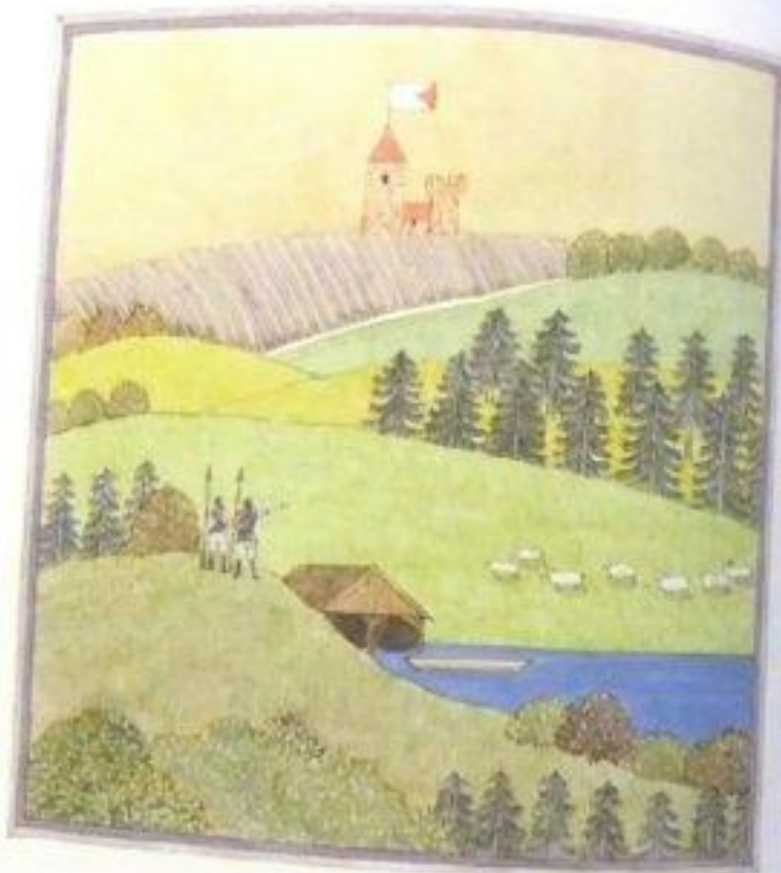
$$5! = 120 = 1 \times 2 \times 3 \times 4 \times 5$$

	A	B	C	D
1	0	!	=	1
2	1	!	=	1
3	2	!	=	2
4	3	!	=	6
5	4	!	=	24
6	5	!	=	120
7	6	!	=	720
8	7	!	=	5040
9	8	!	=	40320
10	9	!	=	362880
11	10	!	=	3628800
12	11	!	=	39916800
13	12	!	=	479001600

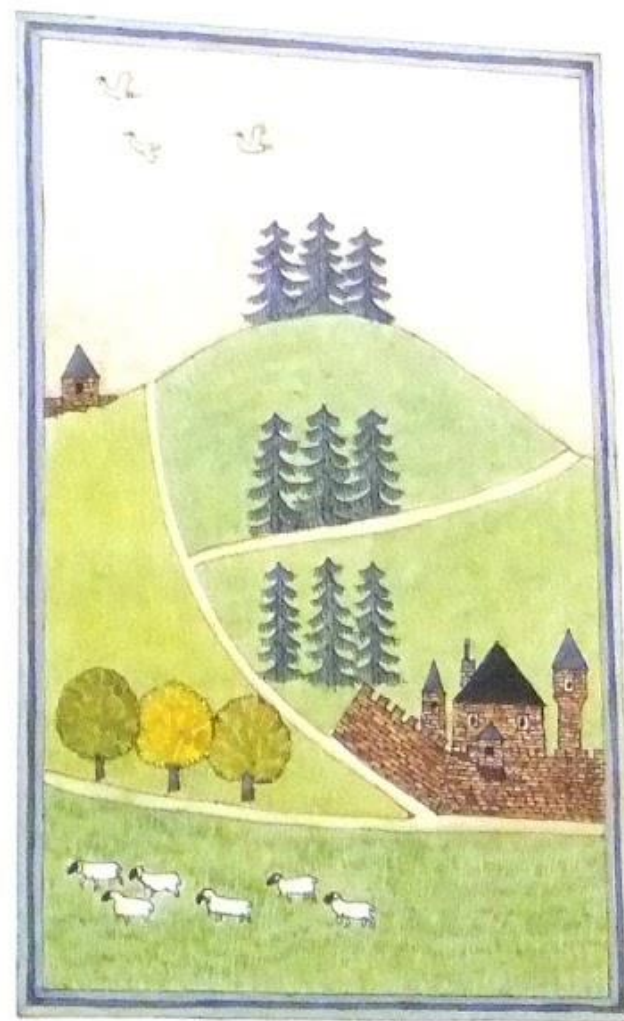
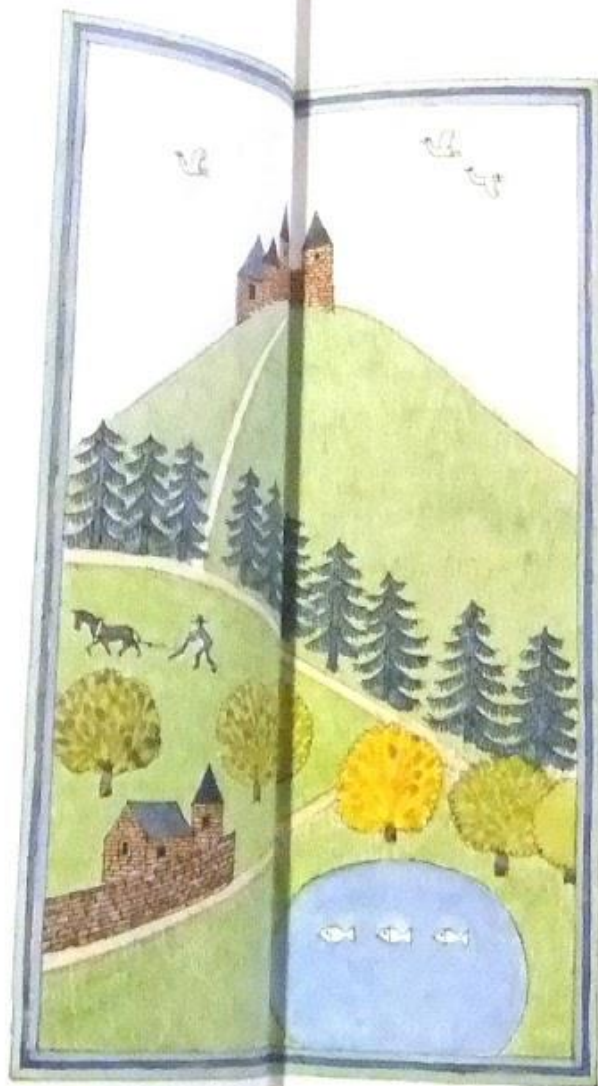
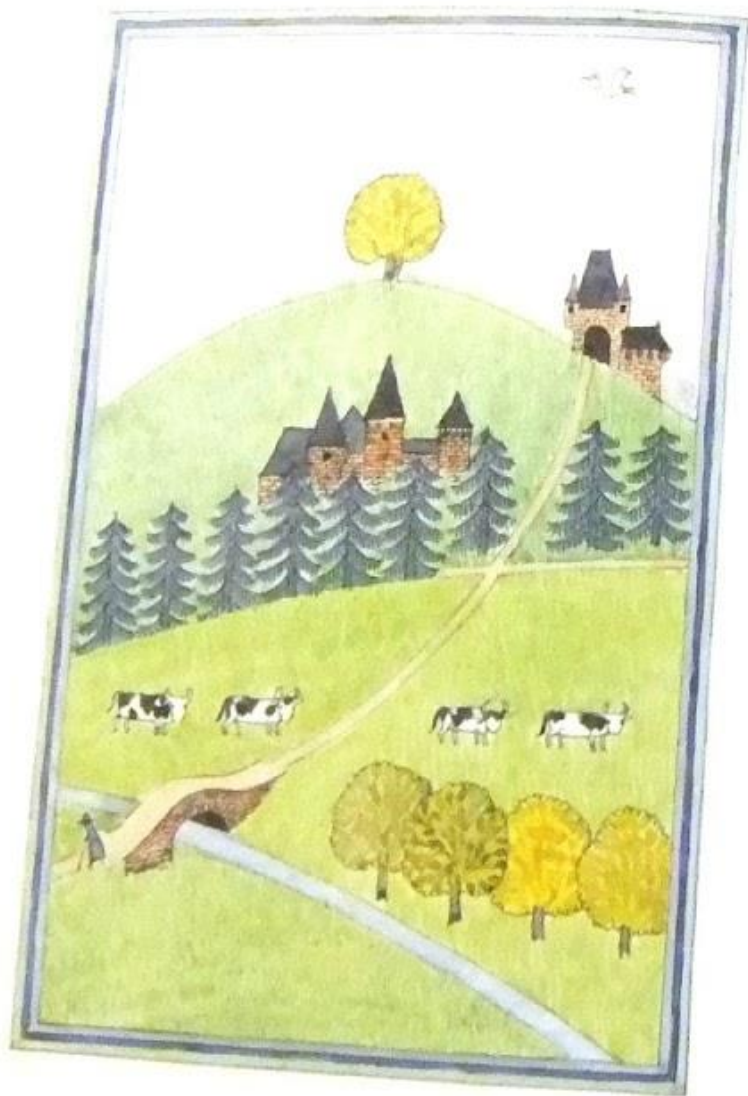
- Factorials grow very quickly.
- This is why counting things by hand can be difficult.



On the sea was 1 island.



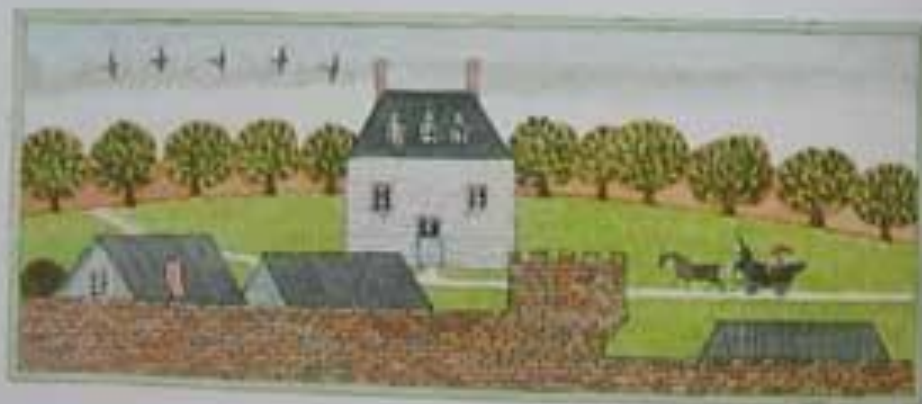
On the island there were 2 countries.



Within each country there were 3 mountains.

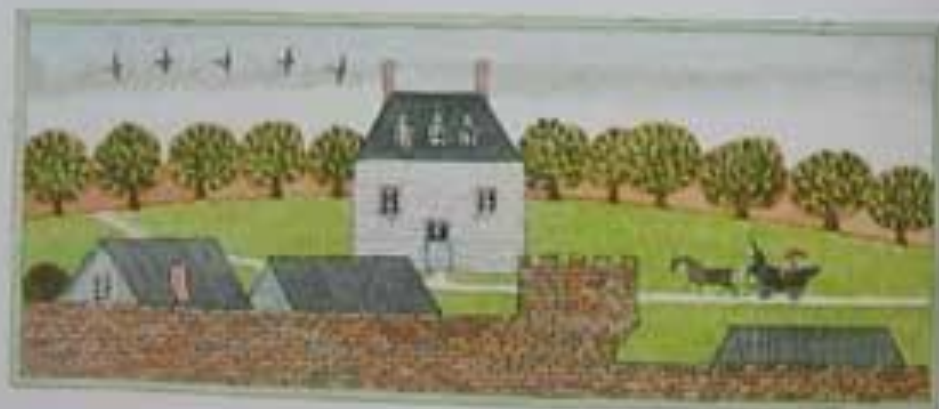


Within each walled kingdom there were 5 villages.





Within each walled kingdom there were 5 villages.





to each corner there were 8 cupboards.



Here is a list of the factorials in this story:

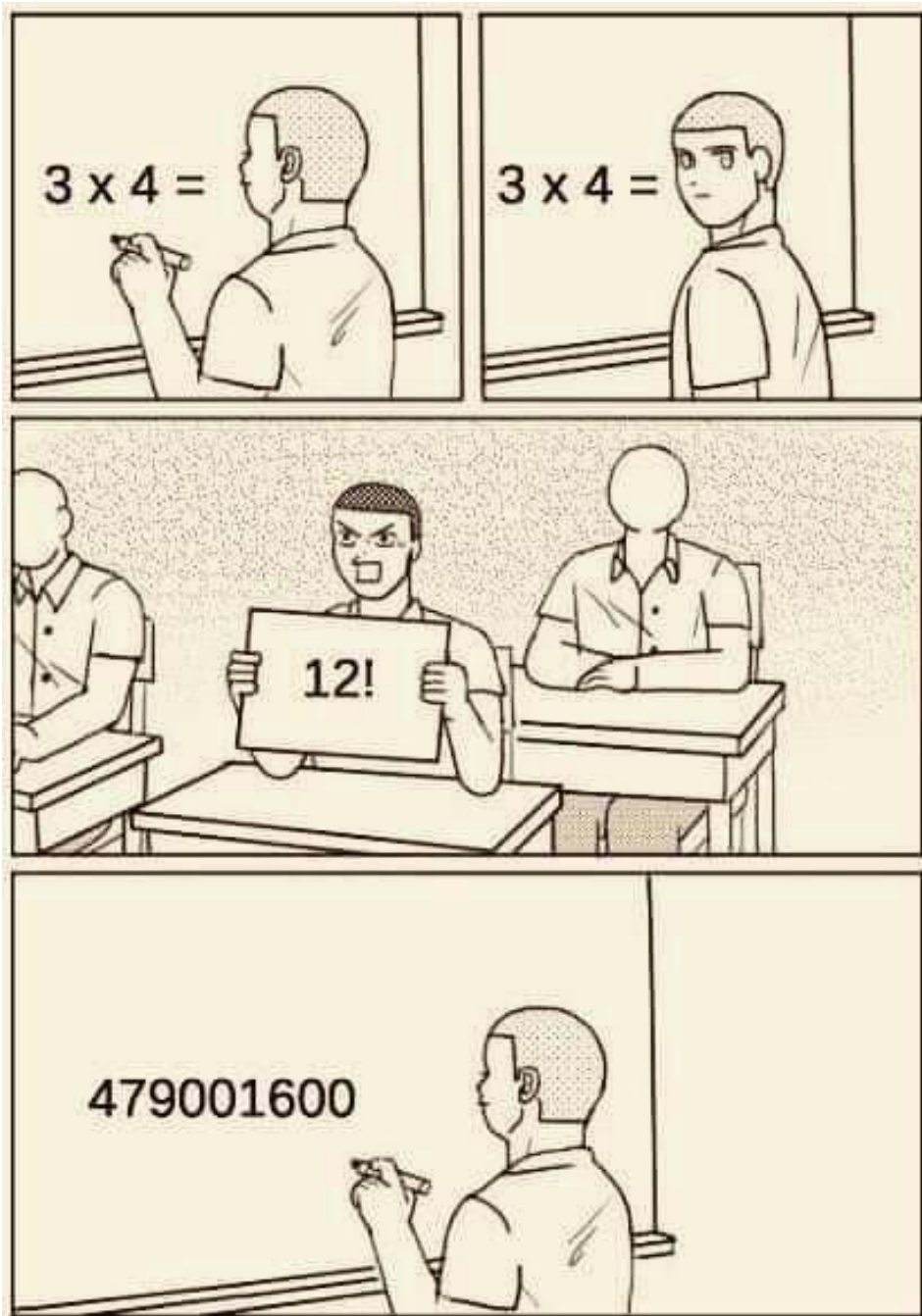
$1! = 1 \times 1! =$	$1 =$	1 (island)
$2! = 2 \times 1! =$	$1 \times 2 =$	2 (countries)
$3! = 3 \times 2! =$	$1 \times 2 \times 3 =$	6 (mountains)
$4! = 4 \times 3! =$	$1 \times 2 \times 3 \times 4 =$	24 (walled kingdoms)
$5! = 5 \times 4! =$	$1 \times 2 \times 3 \times 4 \times 5 =$	120 (villages)
$6! = 6 \times 5! =$	$1 \times 2 \times 3 \times 4 \times 5 \times 6 =$	720 (houses)
$7! = 7 \times 6! =$	$1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 =$	5,040 (rooms)
$8! = 8 \times 7! =$	$1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 =$	40,320 (cupboards)
$9! = 9 \times 8! =$	$1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9 =$	362,880 (boxes)
$10! = 10 \times 9! =$	$1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9 \times 10 =$	3,628,800 (jars)



© 2011 by Snuggly, Inc.

"THIS DECISION IS SO HARD!"

Snuggly
Illustration



Get it?

	A	B	C	D
1	0 !	=		1
2	1 !	=		1
3	2 !	=		2
4	3 !	=		6
5	4 !	=		24
6	5 !	=		120
7	6 !	=		720
8	7 !	=		5040
9	8 !	=		40320
10	9 !	=		362880
11	10 !	=		3628800
12	11 !	=		39916800
13	12 !	=		479001600

How else can you express 5!

How else can you express 5!

$$5! = 1 \times 2 \times 3 \times 4 \times 5$$

How else can you express 5!

$$5! = 1 \times 2 \times 3 \times 4 \times 5$$

$$5! = 4! \times 5$$

$$5! = 1 \times 2 \times 3 \times 4 \times 5$$

How else can you express 5!

$$5! = 1 \times 2 \times 3 \times 4 \times 5$$

$$5! = 4! \times 5$$

$$5! = 1 \times 2 \times 3 \times 4 \times 5$$

$$5! = 3! \times 4 \times 5$$

Write using factorials

$$3 \times 4 \times 5 \times 6 =$$

Write using factorials

$$3 \times 4 \times 5 \times 6 = \frac{1 \times 2 \times 3 \times 4 \times 5 \times 6}{1 \times 2}$$

Write using factorials

$$3 \times 4 \times 5 \times 6 = \frac{1 \times 2 \times 3 \times 4 \times 5 \times 6}{1 \times 2}$$
$$= \frac{6!}{2!}$$

A
"Chopped"
off
factorial.

Simplify, then evaluate

$$\frac{7!}{4!} =$$

Simplify, then evaluate

$$\frac{7!}{4!} = \frac{7 \times 6 \times 5 \times 4!}{4!}$$

Simplify, then evaluate

$$\begin{aligned}\frac{7!}{4!} &= \frac{7 \times 6 \times 5 \times 4!}{4!} \\ &= 7 \times 6 \times 5\end{aligned}$$

Simplify, then evaluate


$$\frac{7!}{4!} = \frac{7 \times 6 \times 5 \times 4!}{4!}$$

$$= 7 \times 6 \times 5$$

$$= 210$$

Factorials:


Question	Calculation	Answer
1!	1	1
2!	2 x 1	2
3!	3 x 2 x 1	6
4!	4 x 3 x 2 x 1	24
5!	5 x 4 x 3 x 2 x 1	120
6!	6 x 5 x 4 x 3 x 2 x 1	720



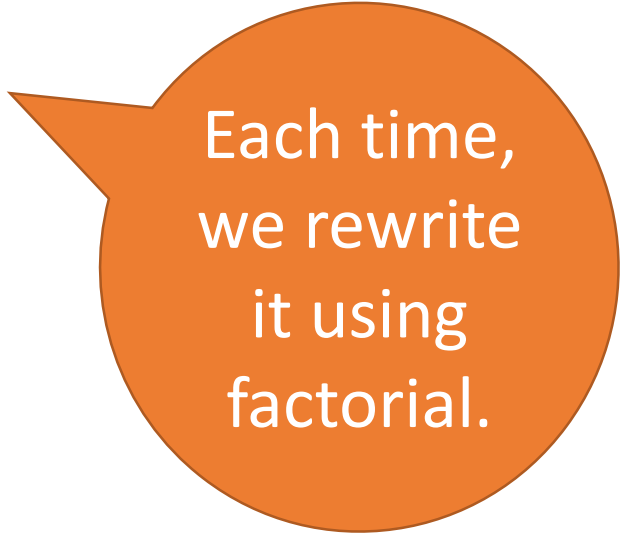
Now we
journey back
into the land
of comp sci.

It's like a loop:

n!	Values
5!	5 x 4!
	5 x 4 x 3!
	5 x 4 x 3 x 2!
	5 x 4 x 3 x 2 x 1



Each time,
we
progress
towards 1.



Each time,
we rewrite
it using
factorial.

```
public int factorial (int n) {  
    if (n <= 1)  
        return 1;  
    else  
        return factorial (n - 1) * n;  
}
```

The base
case =
when it
stops.

The recursive
case = calling
itself again,
with a smaller
parameter

$$n! = (n-1)! \times n$$

n!

n:
id: num

onClick: answer

Find

id: result

Enter n,
press find.

How do we code the method call in the Find button?

```
public int factorial (int n) {  
    if (n <= 1)  
        return 1;  
    else  
        return factorial (n - 1) * n;  
}
```

n!

n:
id: num

onClick: answer

Find

id: result

Enter n,
press find.

How do we code the method call in the Find button?

```
public void answer (View view) {  
  
}
```

```
public int factorial (int n) {  
    if (n <= 1)  
        return 1;  
    else  
        return factorial (n - 1) * n;  
}
```

n!

n:
id: num

onClick: answer

Find

id: result

Enter n,
press find.

How do we code the method call in the Find button?

```
public void answer (View view) {  
    EditText num = (EditText) findViewById (R.id.num);  
    TextView result = (TextView) findViewById (R.id.result);  
  
}
```

```
public int factorial (int n) {  
    if (n <= 1)  
        return 1;  
    else  
        return factorial (n - 1) * n;  
}
```

n!

n:
id: num

onClick: answer

Find

id: result

Enter n,
press find.

How do we code the method call in the Find button?

```
public void answer (View view) {  
    EditText num = (EditText) findViewById (R.id.num);  
    TextView result = (TextView) findViewById (R.id.result);  
  
    int n = Integer.parseInt(num.getText().toString());  
  
}
```

```
public int factorial (int n) {  
    if (n <= 1)  
        return 1;  
    else  
        return factorial (n - 1) * n;  
}
```

n!

n:
id: num

onClick: answer

Find

id: result

Enter n,
press find.

How do we code the method call in the Find button?

```
public void answer (View view) {  
    EditText num = (EditText) findViewById (R.id.num);  
    TextView result = (TextView) findViewById (R.id.result);  
  
    int n = Integer.parseInt(num.getText().toString());  
    int answer = factorial(n);  
}
```

```
public int factorial (int n) {  
    if (n <= 1)  
        return 1;  
    else  
        return factorial (n - 1) * n;  
}
```


n!

n:
id: num

onClick: answer

Find

id: result

Enter n,
press find.

How do we code the method call in the Find button?

```
public void answer (View view) {  
    EditText num = (EditText) findViewById (R.id.num);  
    TextView result = (TextView) findViewById (R.id.result);  
  
    int n = Integer.parseInt(num.getText().toString());  
    int answer = factorial(n);  
    result.setText(n+ "!=" +answer);  
}
```

```
public int factorial (int n) {  
    if (n <= 1)  
        return 1;  
    else  
        return factorial (n - 1) * n;  
}
```

A brief journey into Fibonacci

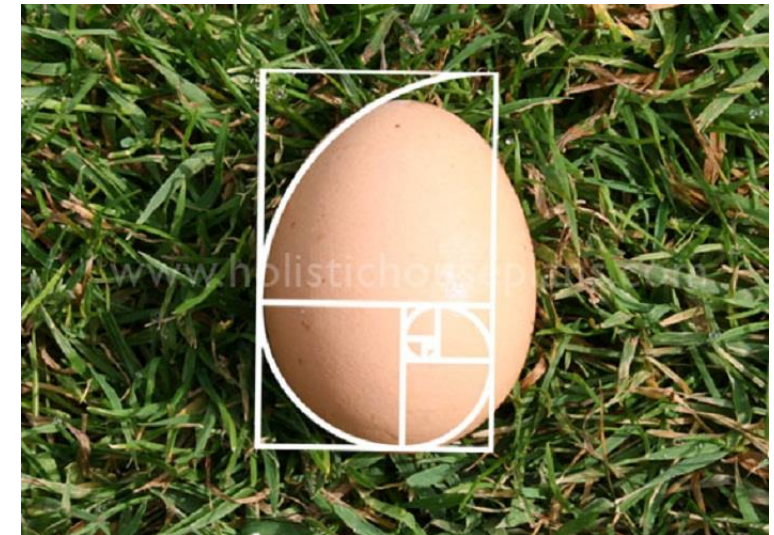
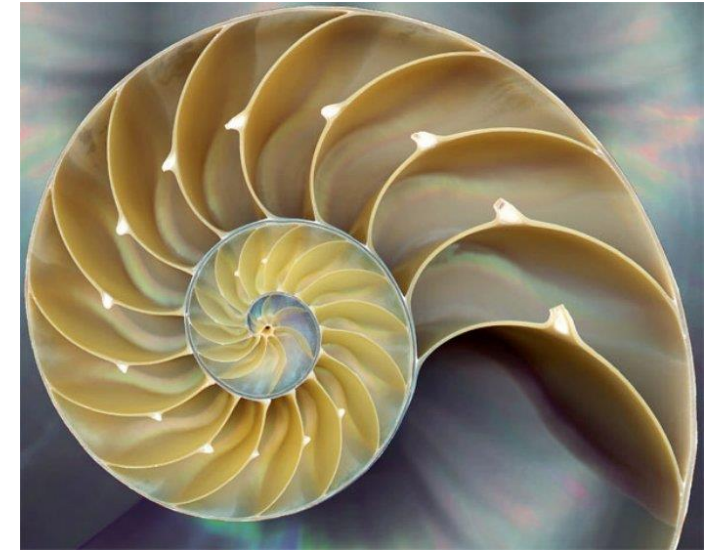
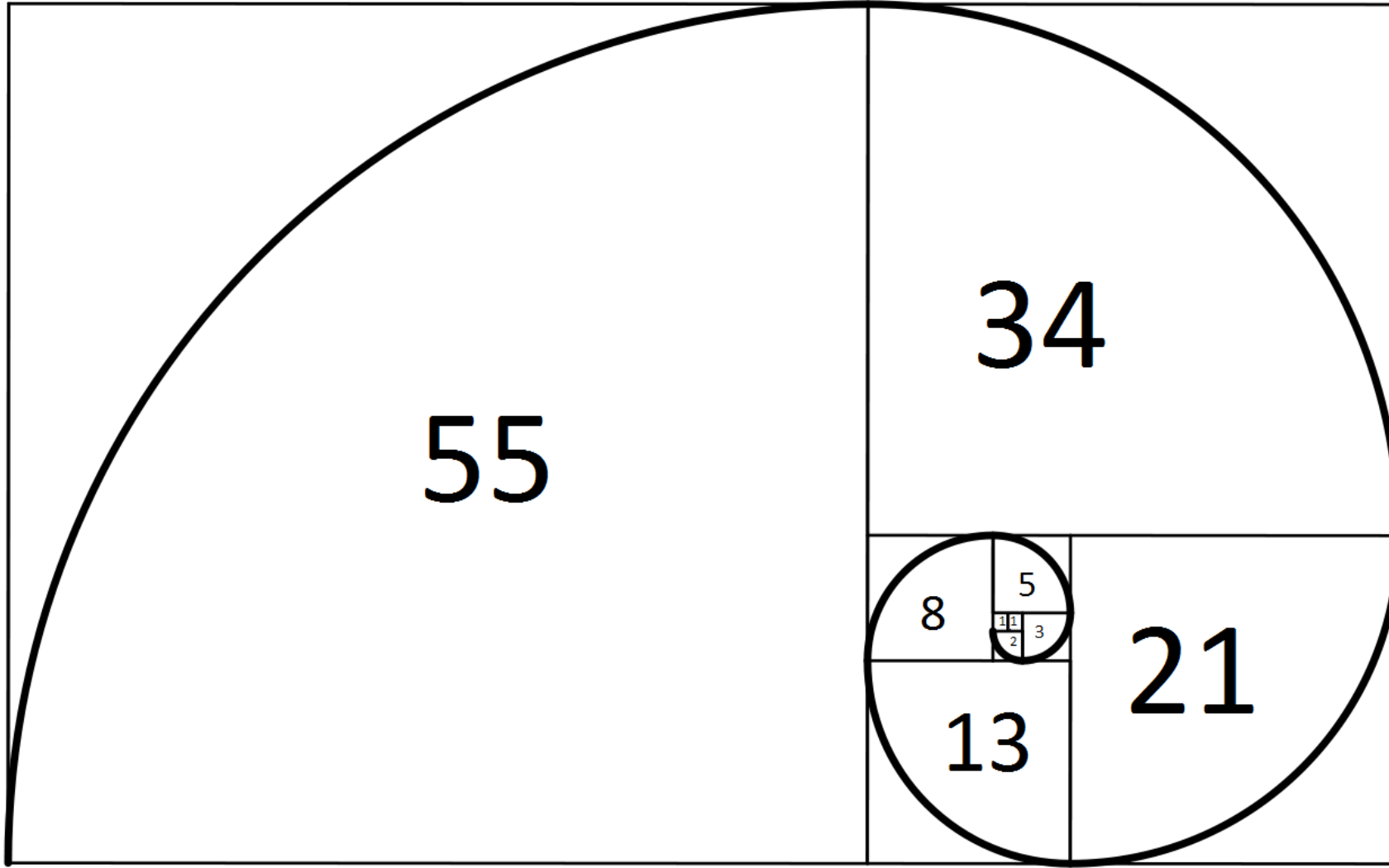
Ahhh... Art

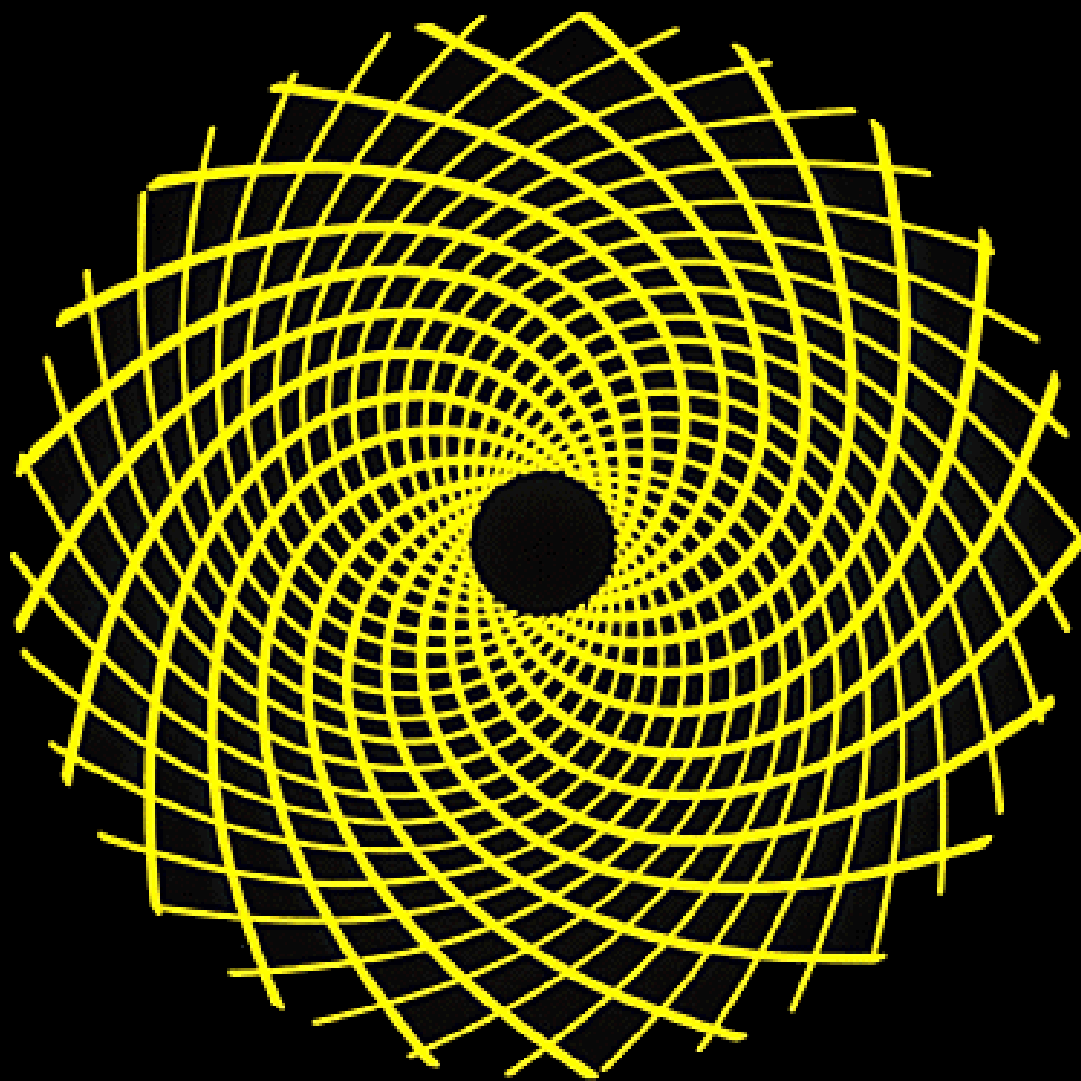
1	2	3	4	5	6	7	8	9	10	11	...
1	1	2	3	5	8	13	21	34	55	89	...

What does fib(6) return?

What does fib(1) return?

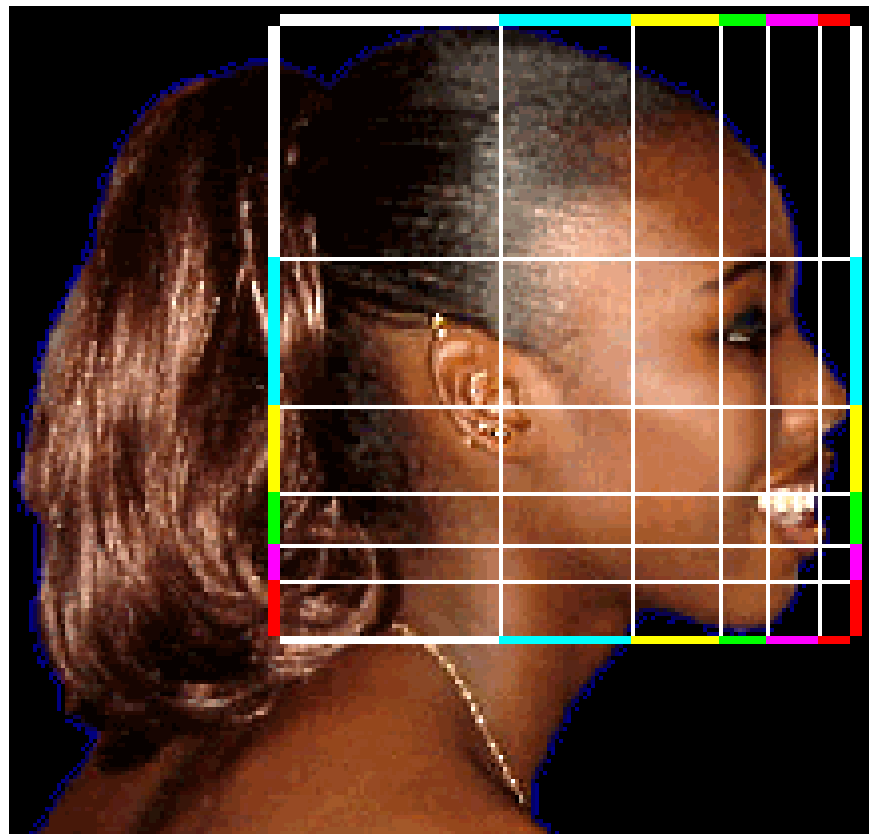
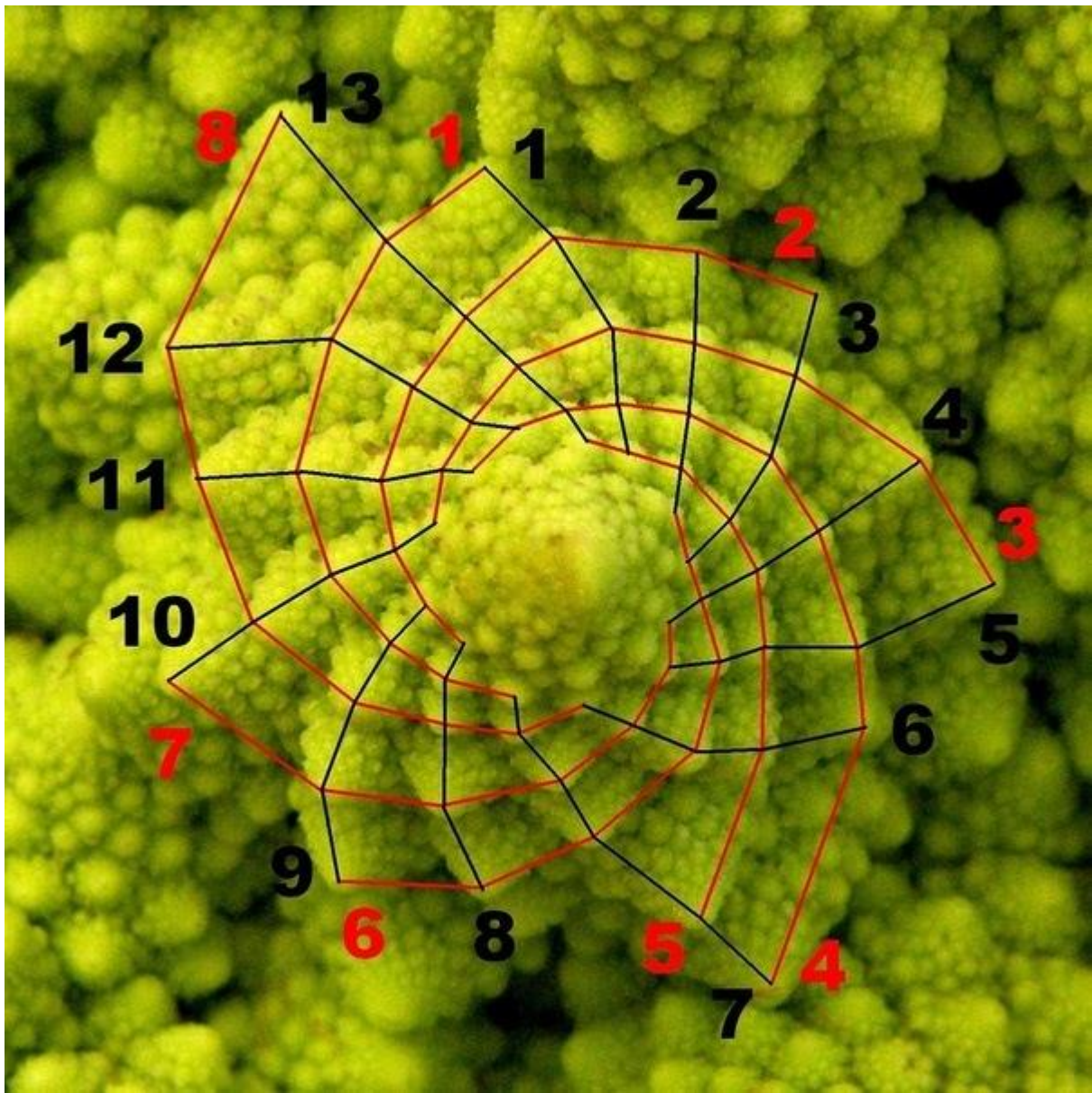
What does fib(9) return?

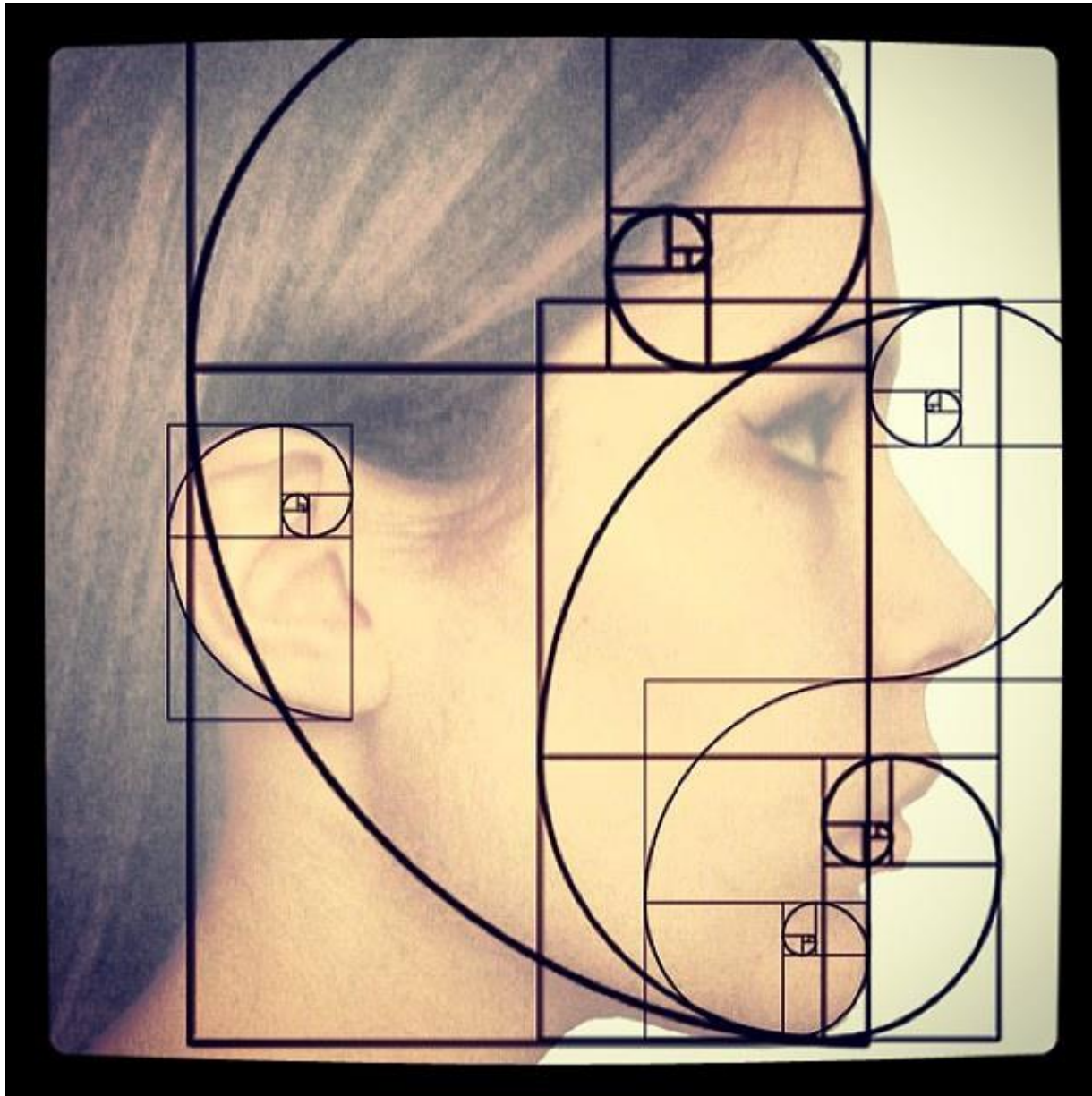




21







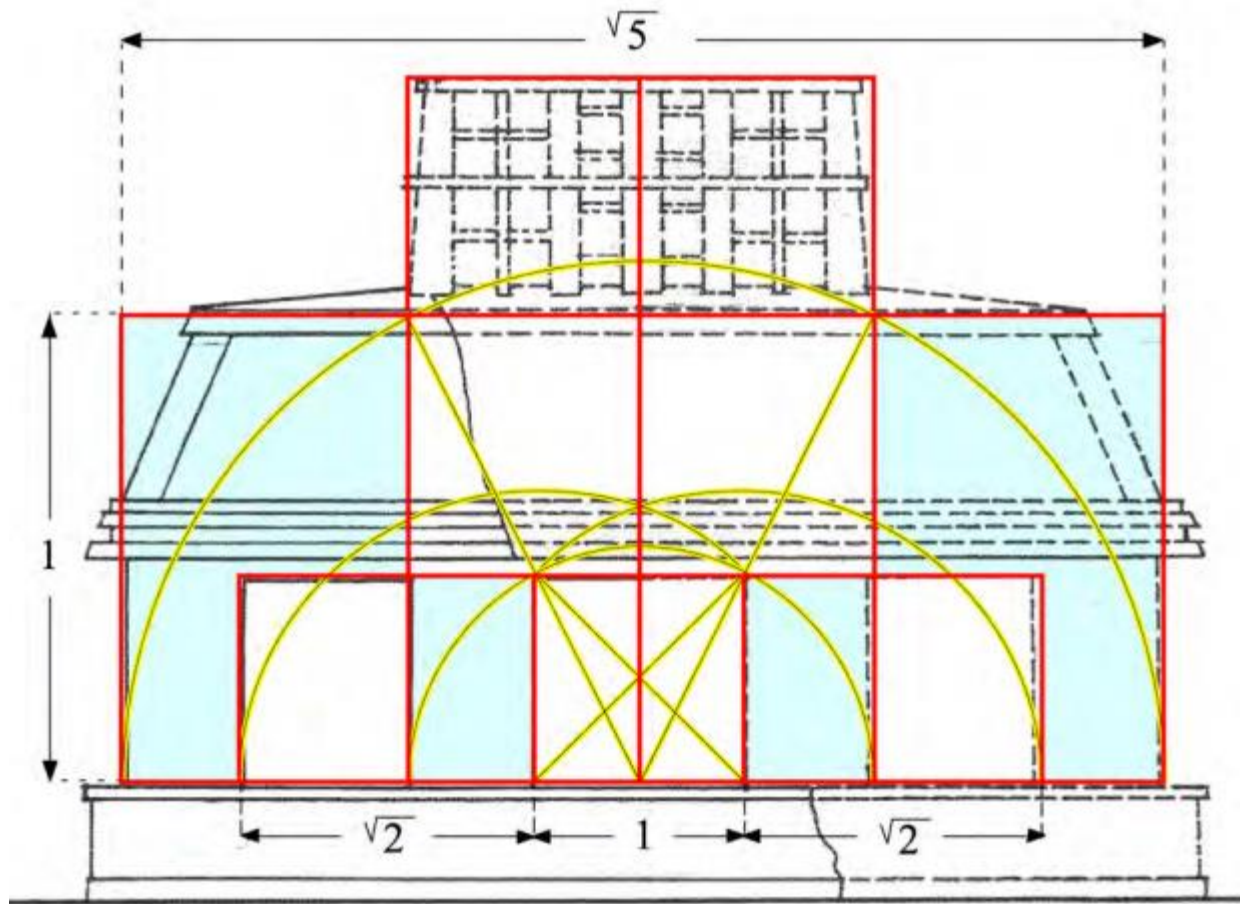
54 mm

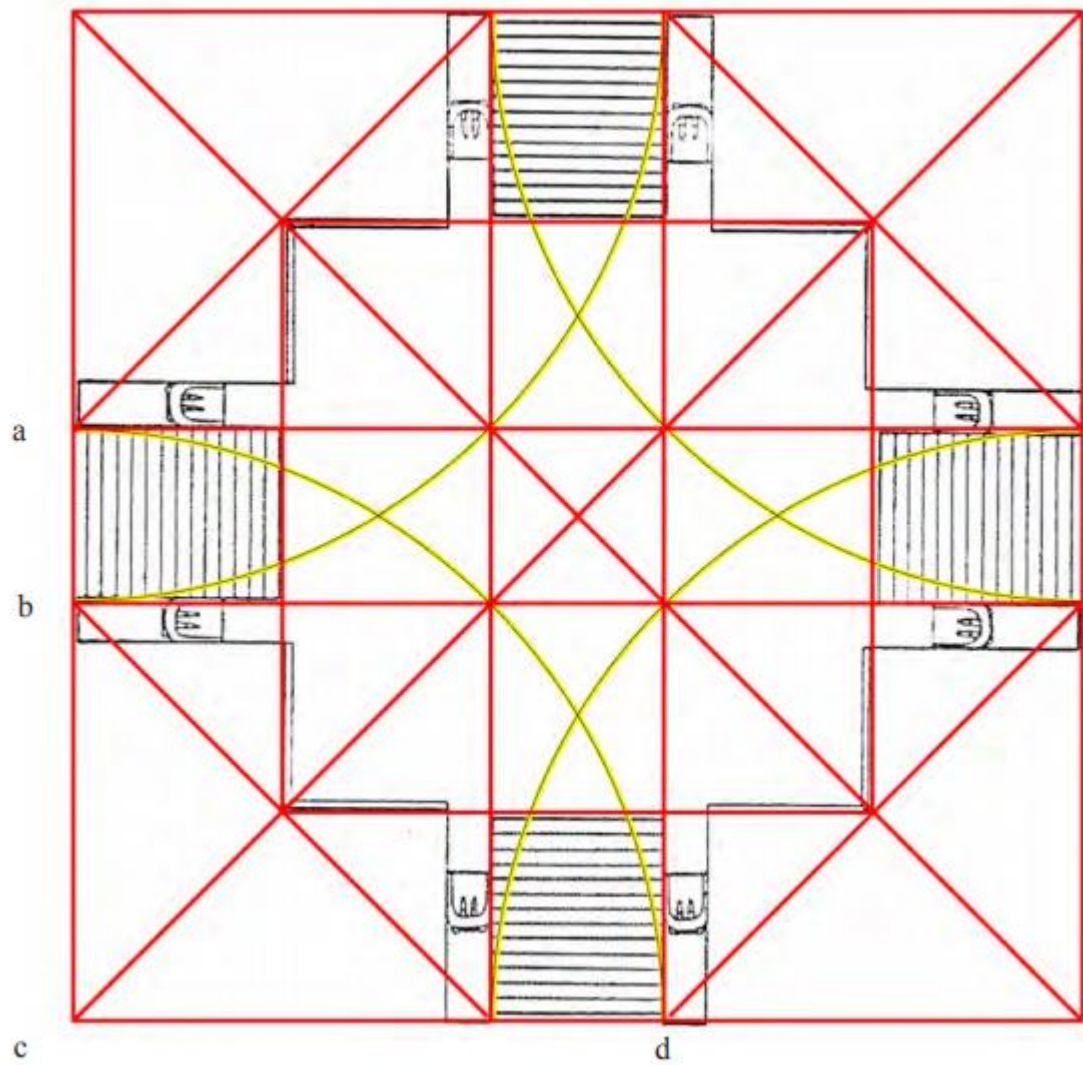
86 mm

$$54/86 = .6279$$

$$\text{Golden Ratio} = .618$$

Elevation of Temple XII,
Palenque, Chiapas, Mexico
Measured Drawing by G. F. Andrews (1974)





Platform of
Venus, Chichen
Itza



If the Fibonacci number is in position 1 or 2,

It is one

Otherwise

It is the sum of two previous Fibonacci numbers

```
public int fib (int n) {  
    if (n == 1 || n == 2)  
        return 1;  
    else  
        return fib (n - 1) + fib (n - 2);  
}
```


```
System.out.println(fib(5));
```

```
fib(5)
```

1	2	3	4	5	6	7	8	9	10	11	...
1	1	2	3	5	8	13	21	34	55	89	...

```
public int fib (int n) {  
    if (n == 1 || n == 2)  
        return 1;  
    else  
        return fib (n - 1) + fib (n - 2);  
}
```

```
System.out.println(fib(5));
```

fib(5)

fib(4) + fib(3)

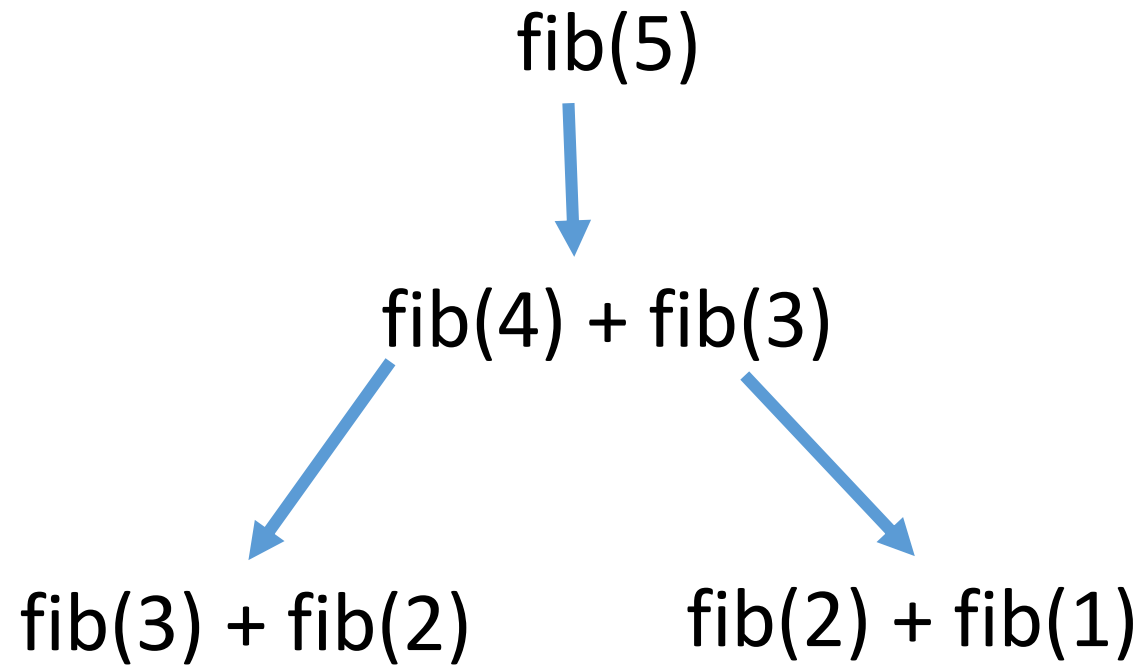
1	2	3	4	5	6	7	8	9	10	11	...
1	1	2	3	5	8	13	21	34	55	89	...

```
public int fib (int n) {  
    if (n == 1 || n == 2)  
        return 1;  
    else  
        return fib (n - 1) + fib (n - 2);  
}
```



```
System.out.println(fib(5));
```

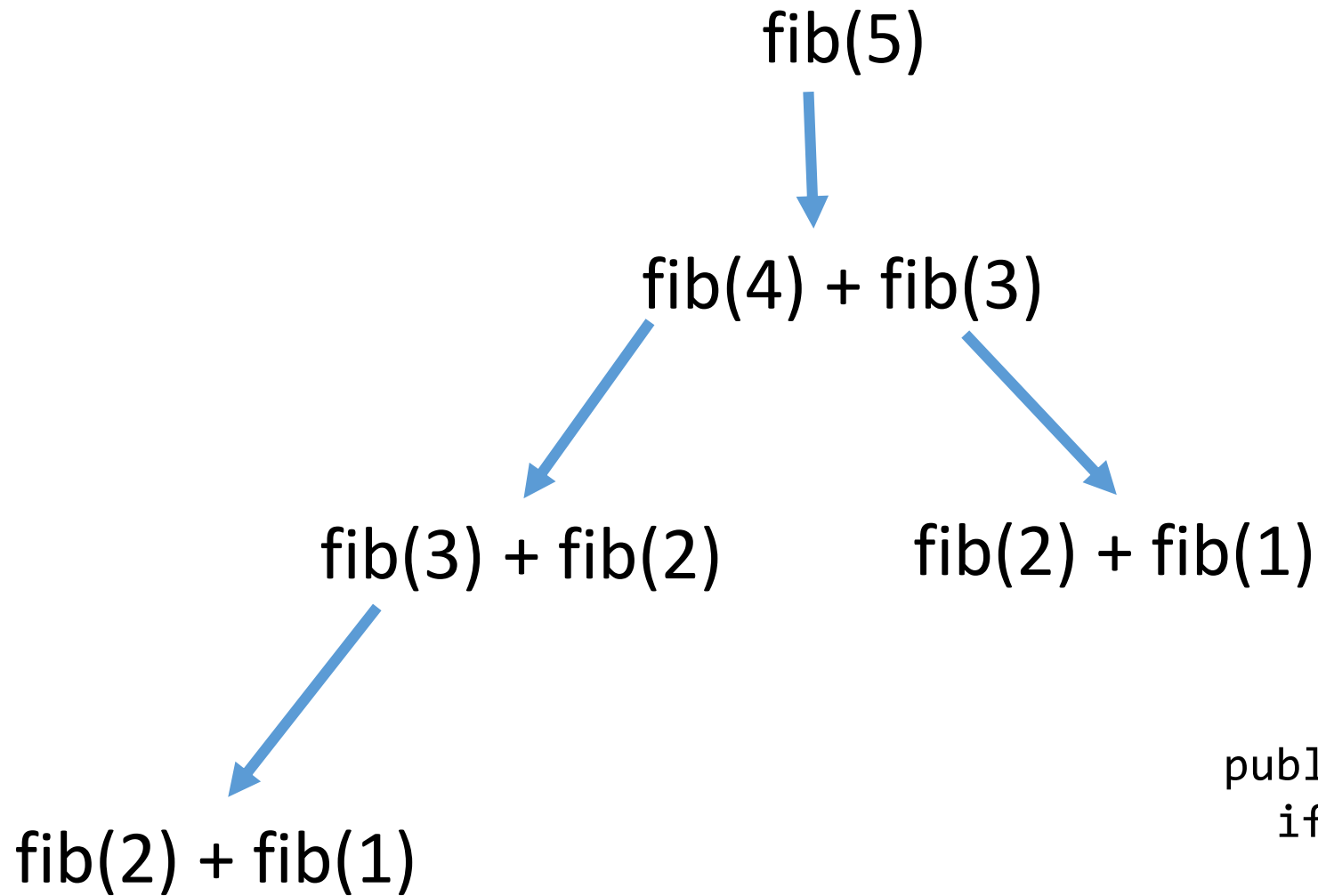
1	2	3	4	5	6	7	8	9	10	11	...
1	1	2	3	5	8	13	21	34	55	89	...



```
public int fib (int n) {  
    if (n == 1 || n == 2)  
        return 1;  
    else  
        return fib (n - 1) + fib (n - 2);  
}
```

```
System.out.println(fib(5));
```

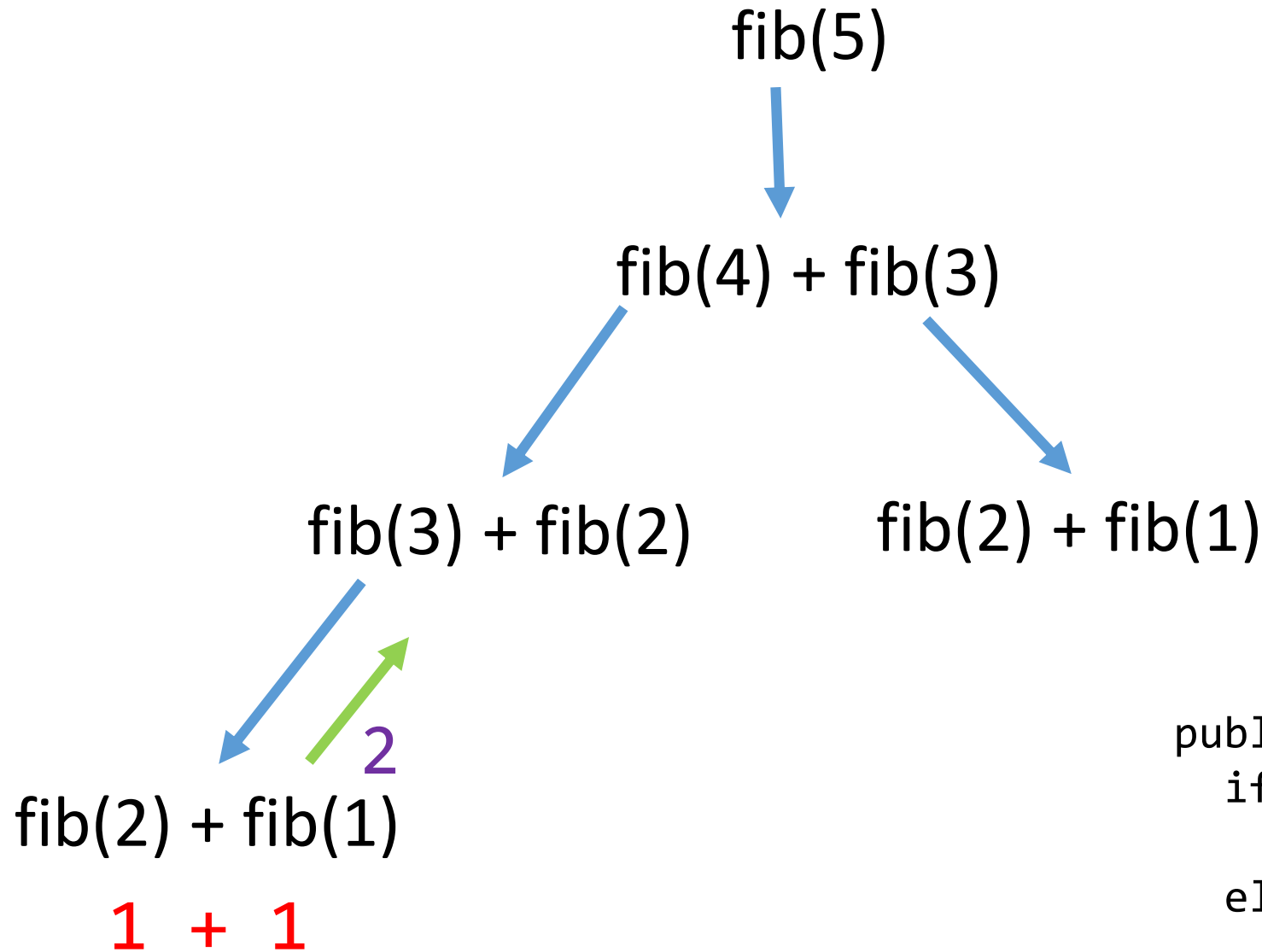
1	2	3	4	5	6	7	8	9	10	11	...
1	1	2	3	5	8	13	21	34	55	89	...



```
public int fib (int n) {  
    if (n == 1 || n == 2)  
        return 1;  
    else  
        return fib (n - 1) + fib (n - 2);  
}
```

```
System.out.println(fib(5));
```

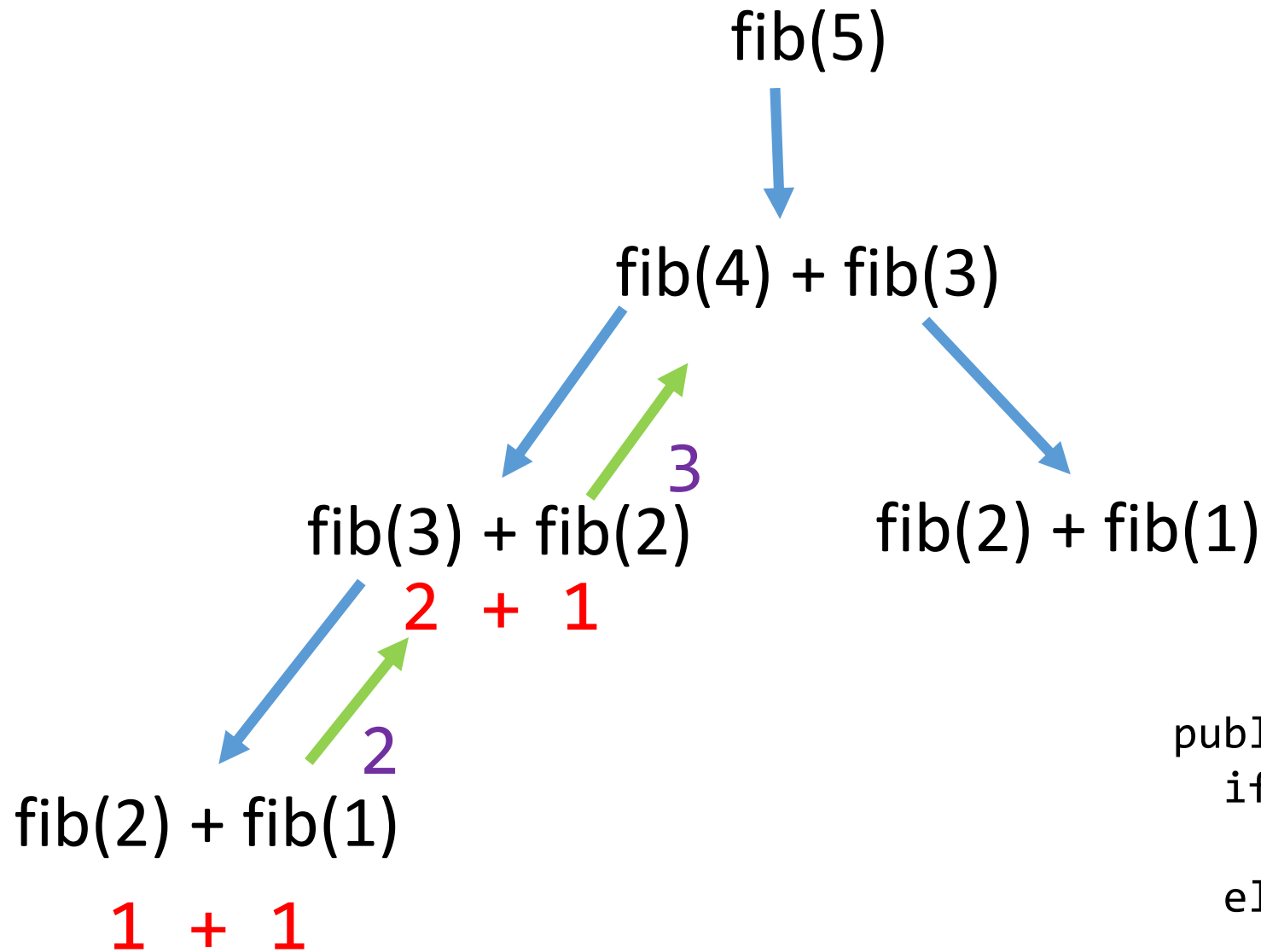
1	2	3	4	5	6	7	8	9	10	11	...
1	1	2	3	5	8	13	21	34	55	89	...



```
public int fib (int n) {  
    if (n == 1 || n == 2)  
        return 1;  
    else  
        return fib (n - 1) + fib (n - 2);  
}
```

```
System.out.println(fib(5));
```

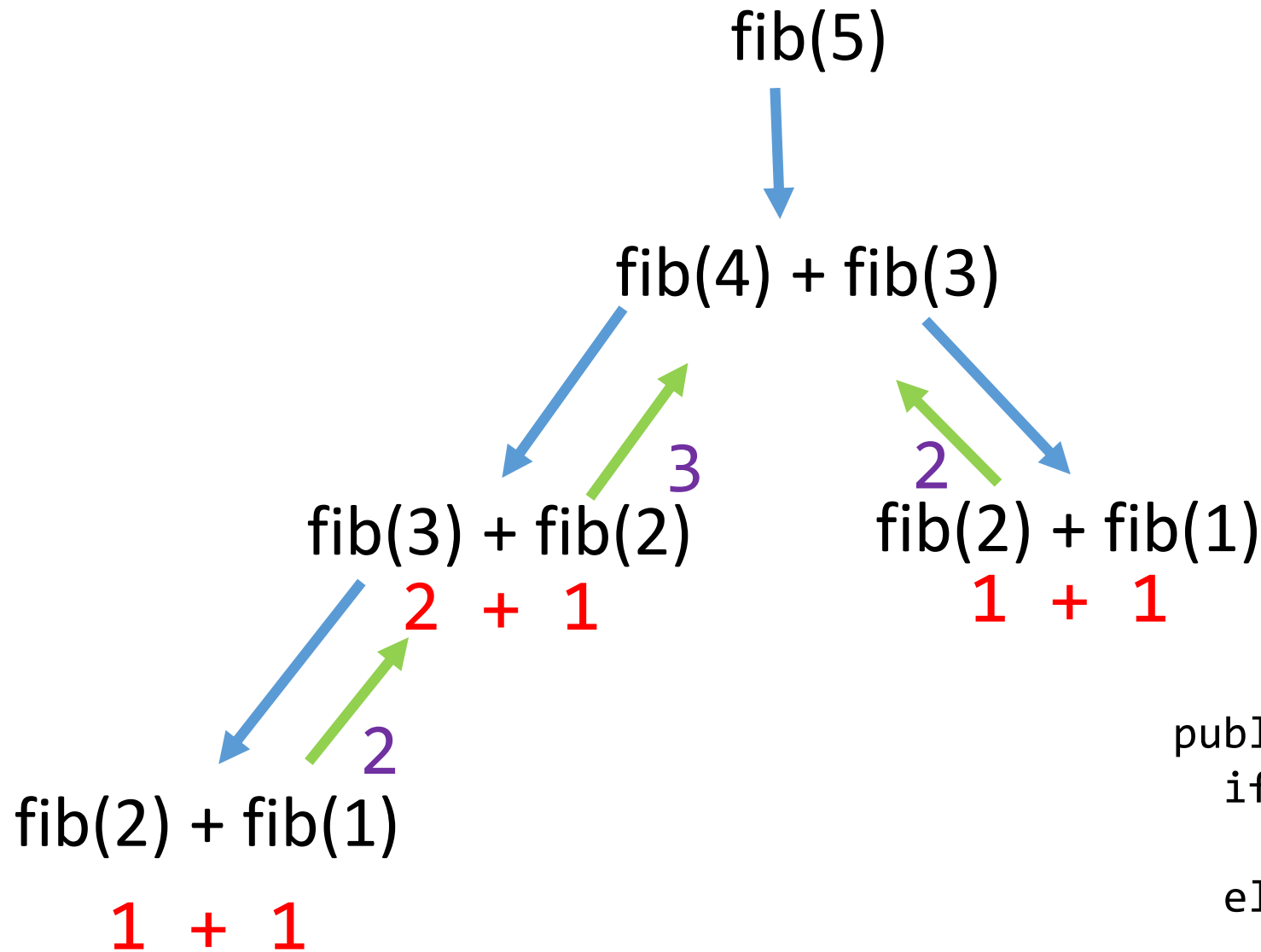
1	2	3	4	5	6	7	8	9	10	11	...
1	1	2	3	5	8	13	21	34	55	89	...



```
public int fib (int n) {  
    if (n == 1 || n == 2)  
        return 1;  
    else  
        return fib (n - 1) + fib (n - 2);  
}
```

```
System.out.println(fib(5));
```

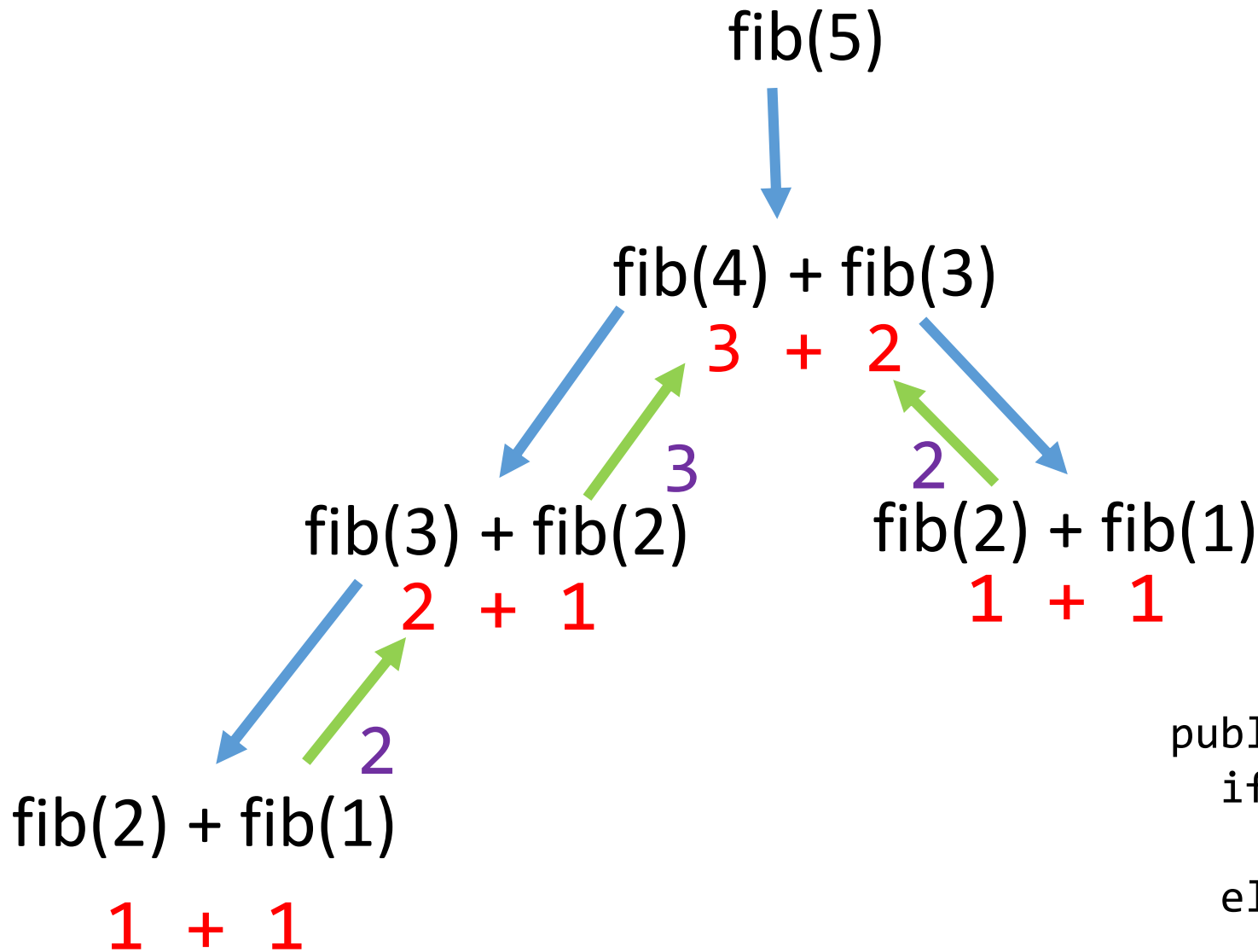
1	2	3	4	5	6	7	8	9	10	11	...
1	1	2	3	5	8	13	21	34	55	89	...



```
public int fib (int n) {  
    if (n == 1 || n == 2)  
        return 1;  
    else  
        return fib (n - 1) + fib (n - 2);  
}
```

```
System.out.println(fib(5));
```

1	2	3	4	5	6	7	8	9	10	11	...
1	1	2	3	5	8	13	21	34	55	89	...

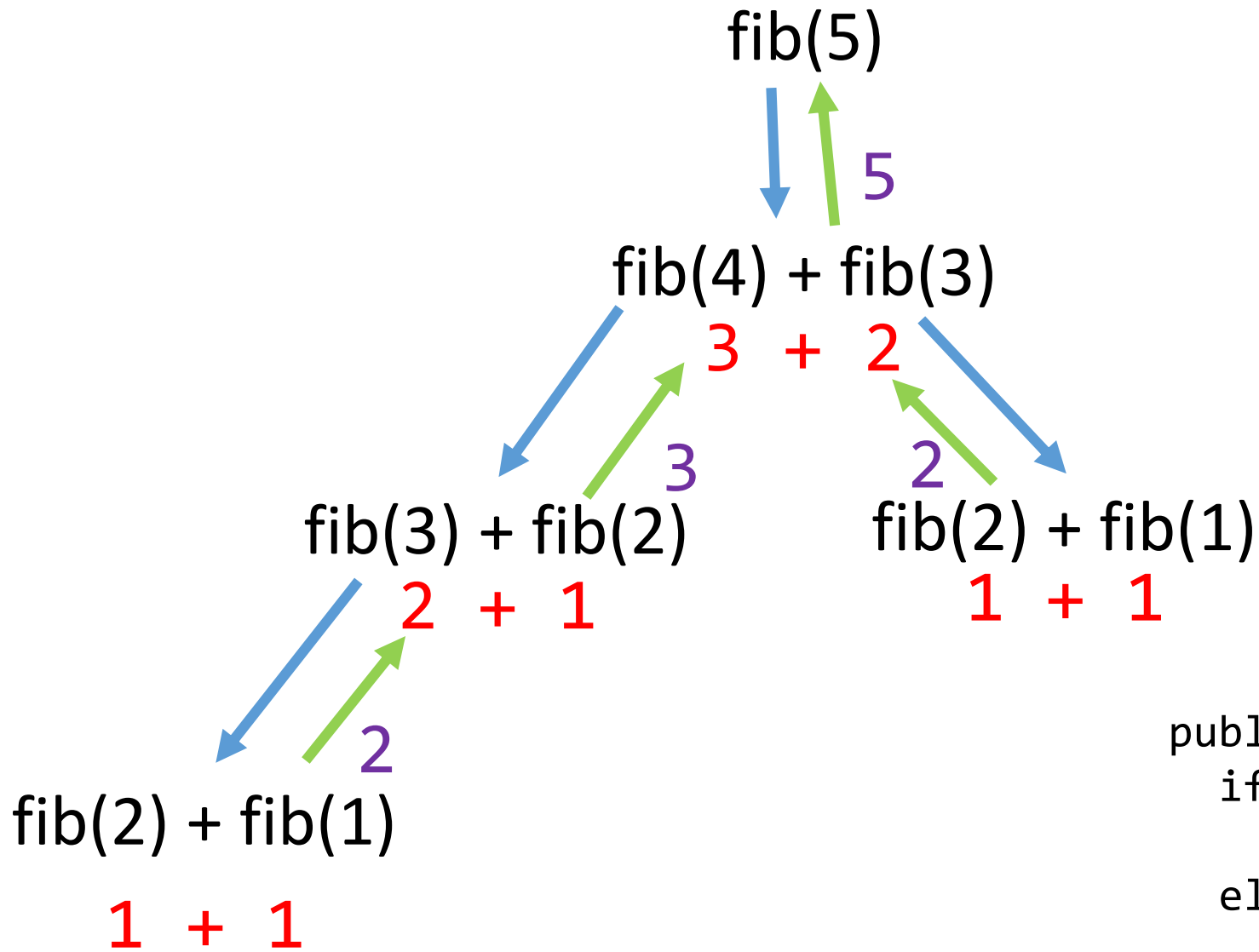


```
public int fib (int n) {  
    if (n == 1 || n == 2)  
        return 1;  
    else  
        return fib (n - 1) + fib (n - 2);  
}
```

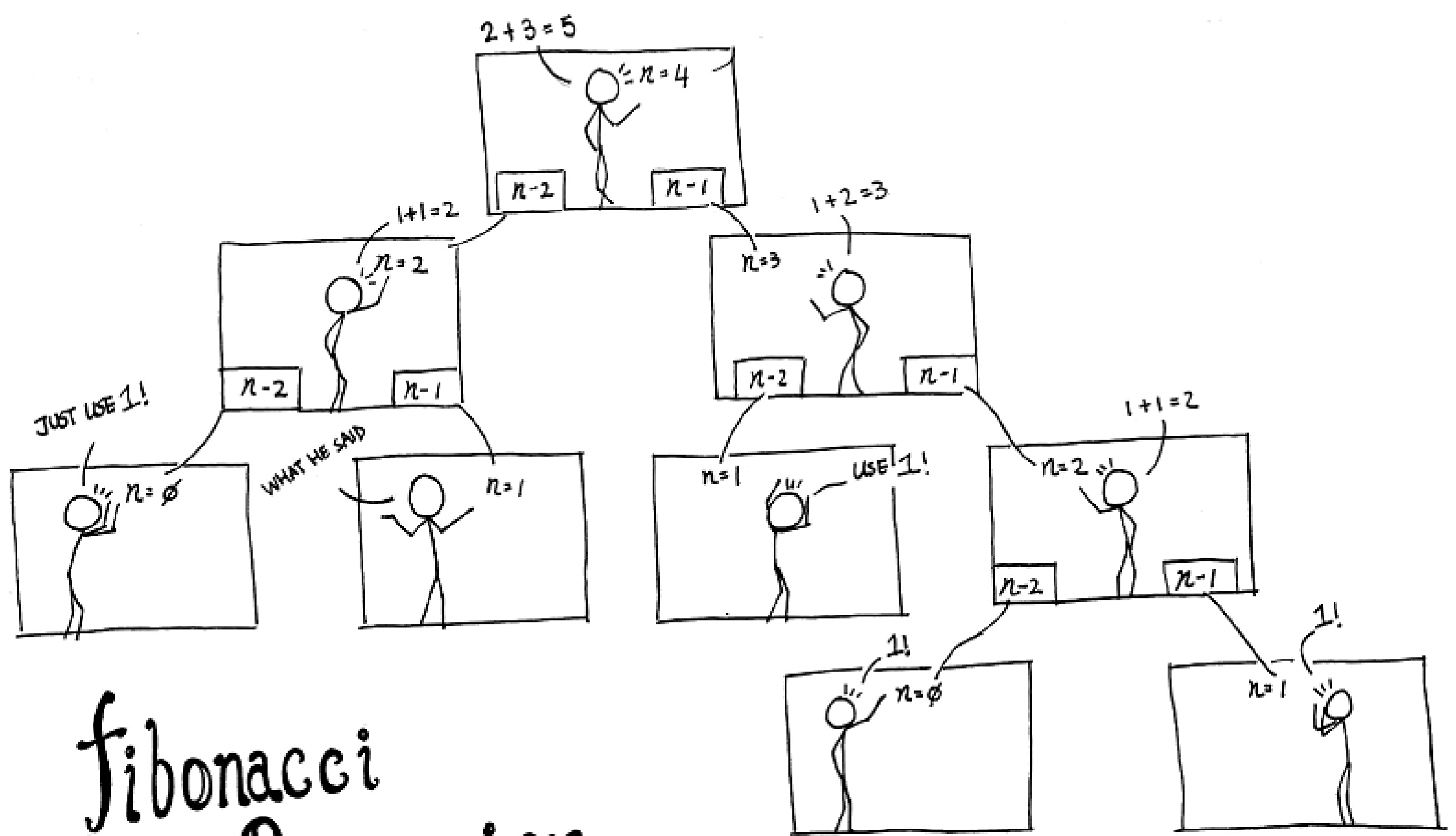


```
System.out.println(fib(5));
```

1	2	3	4	5	6	7	8	9	10	11	...
1	1	2	3	5	8	13	21	34	55	89	...



```
public int fib (int n) {  
    if (n == 1 || n == 2)  
        return 1;  
    else  
        return fib (n - 1) + fib (n - 2);  
}
```



Fibonacci Recursion

DL 2017

TO-DO LIST

1. Make a to-do list

Recursion

- A method that calls itself.
- The base case is the starting point (or stopping point depending on perspective). It is the answer we know.
- The recursive base is the part of the code where the method calls itself. In the recursive case, the parameter progresses towards the stopping condition.
- It must have an if, so that you can have a base case and a recursive case.
- Recursion is useful for things that can be defined in terms of themselves. For example, the term in a Fibonacci sequence is the previous two terms in the Fibonacci sequence added together.

Recursion vs Loops

- All recursive code can be replaced by a loop and vice versa.
- Loops are better for coding sequences of values – they are more efficient.
- Loops are also generally easier to understand and code.
- Recursion is better for sorting (Quicksort and Mergesort are recursive).
- Recursion also makes logical sense in situations like searching and the Fibonacci sequence. Those things are easily defined recursively. Their recursive code is very elegant.
- In a loop, we have the loop stopping condition. In recursion, we have a base case.
- In a loop, we progress to the loop stopping condition. In recursion, our parameters get smaller in our recursive case.
- In a loop, we can have an infinite loop. In recursion, we have a stack overflow error.