

Recursion

Non mathematic definitions

The control of a large force is the same principle as the control of a few men: it is merely a question of dividing up their numbers.

— Sun Zi, *The Art of War* (c. 400CE), translated by Lionel Giles (1910)

Our life is frittered away by detail. . . . Simplify, simplify.

— Henry David Thoreau, *Walden* (1854)

Do the hard jobs first. The easy jobs will take care of themselves.

— attributed to Dale Carnegie

Abstract definitions

Recursion - the definition, description, image of an object or process inside this object or process itself, that is, the situation when the object is part of itself. The term "recursion" is used in various special fields of knowledge - from linguistics to logic. Recursion finds the widest application in mathematics and computer science.

A recursive function is a numerical function $f(n)$ of a numerical argument, which contains itself in its record. Such a record allows us to calculate the values of $f(n)$ based on the values of $f(n-1)$, $f(n-2)$, ... similar to reasoning by induction. For the calculation to complete for any n , it is necessary that for some n the function be defined non-recursively (for example, for $n = 1$, $n = 0$;))

Recursion FaQ

Q. What is recursion?

A. When something is specified in terms of itself.

Q. Why learn recursion?

A.:

- Represents a new mode of thinking.
- Provides a powerful programming paradigm.
- Enables reasoning about correctness.
- Gives insight into the nature of computation.

Q. Where recursion is used?

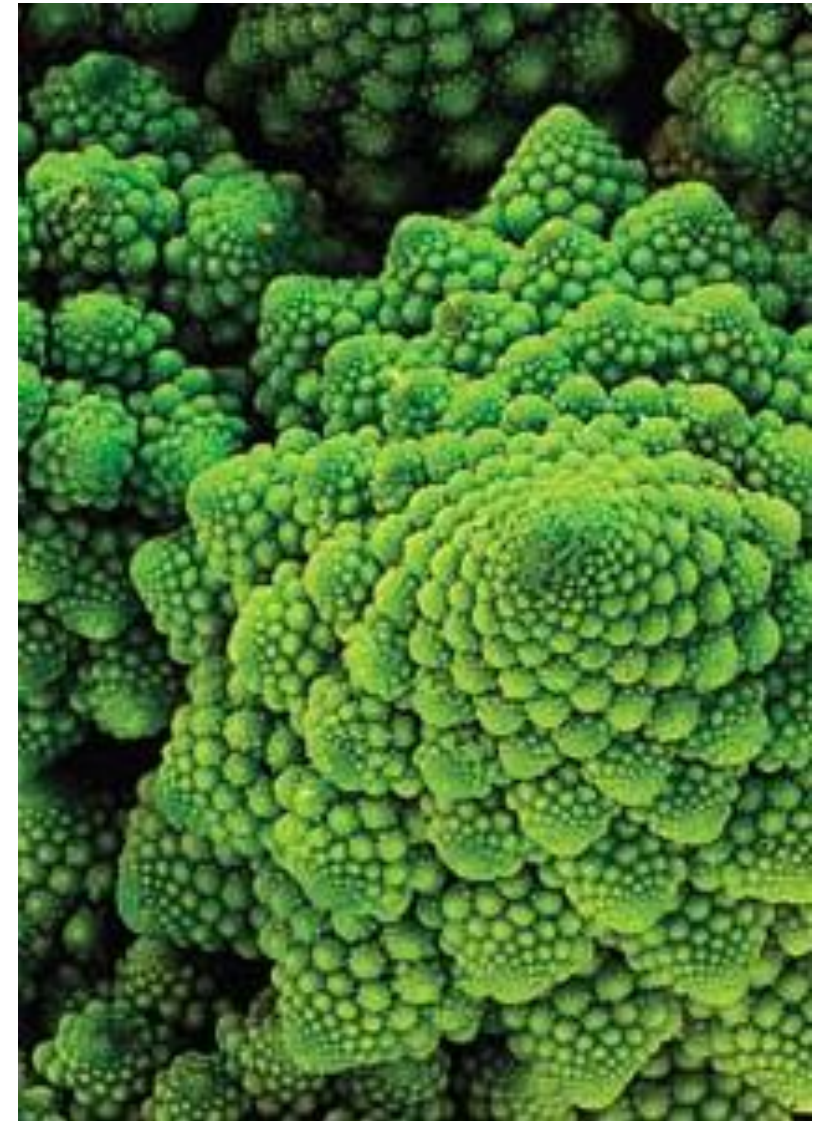
A.

Many computational artifacts are naturally self-referential:

- File system with folders containing folders.
- Fractal graphical patterns.
- Divide-and-conquer algorithms (ex. Tower of Hanoi)

Recursion in Nature

Romanesco cabbage is the most characteristic example of a fractal object in nature. Cabbage buds grow in the form of a certain spiral, which is called logarithmic, and the number of cabbage buds coincides with the Fibonacci number. Fibonacci numbers are elements of a numerical sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946 ... in which each subsequent number is equal to the sum of the two previous numbers. They got their name in honor of the medieval mathematician Leonardo of Pisa (known as Fibonacci). Each part of the elements of Romanesco cabbage has the same shape as the whole head of cabbage. This property is repeated with regularity at various scales.

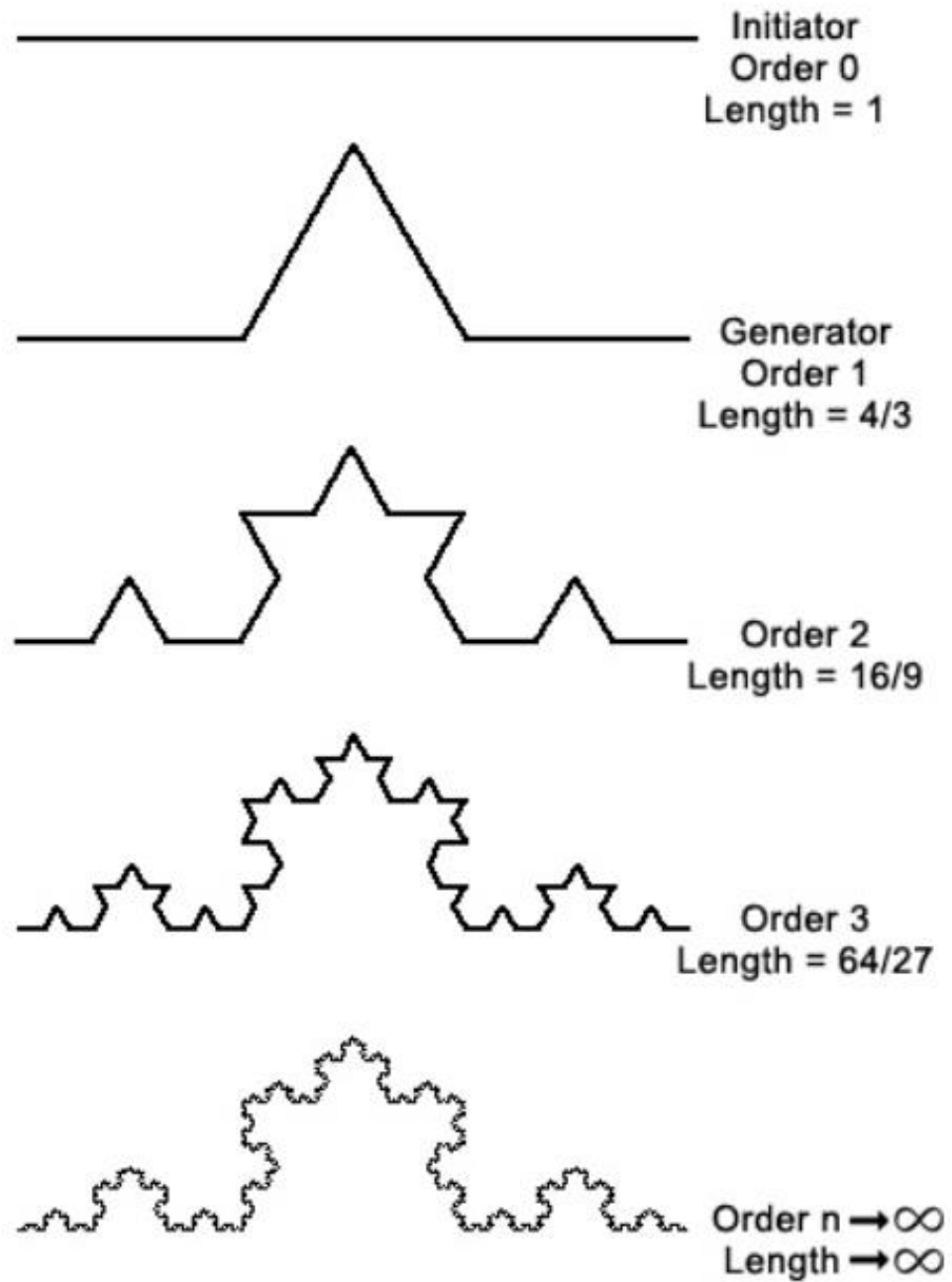




Recursive functions

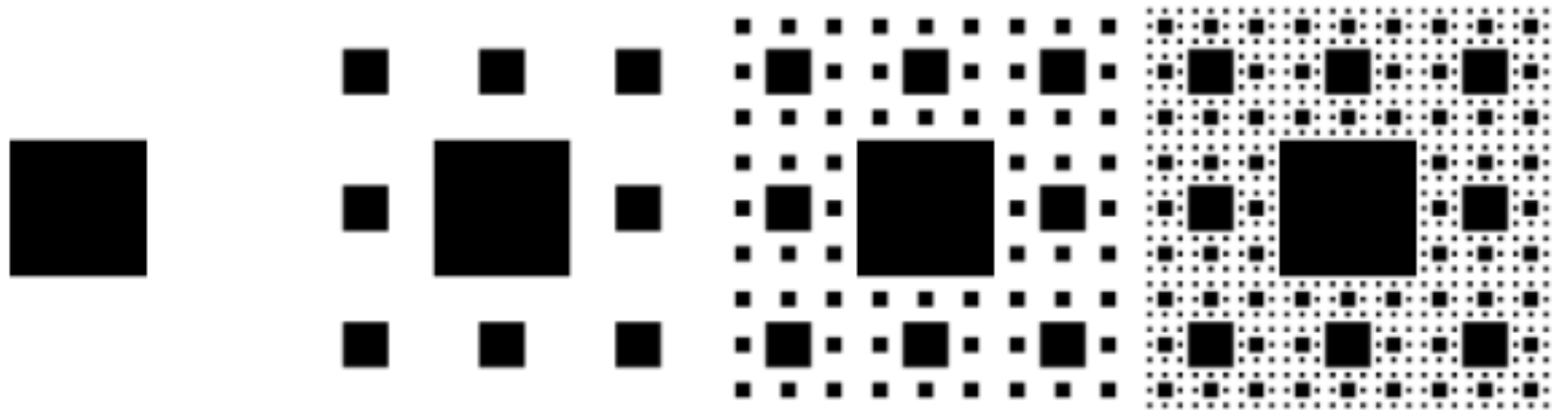
FRACTALS

Koch curve

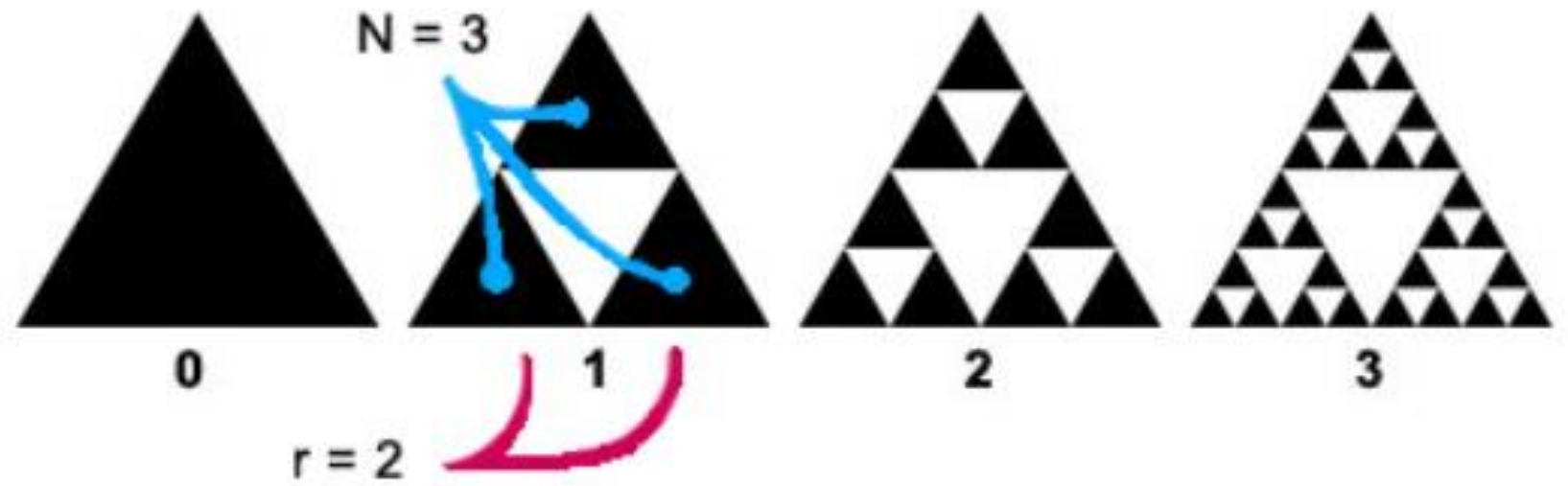


Sierpinski carpet

The construction of the Sierpinski carpet begins with a square. The square is cut into 9 congruent subsquares in a 3-by-3 grid, and the central subsquare is removed. The same procedure is then applied recursively to the remaining 8 subsquares, ad infinitum. It can be realised as the set of points in the unit square whose coordinates written in base three do not both have a digit '1' in the same position.



Sierpinsky triangle





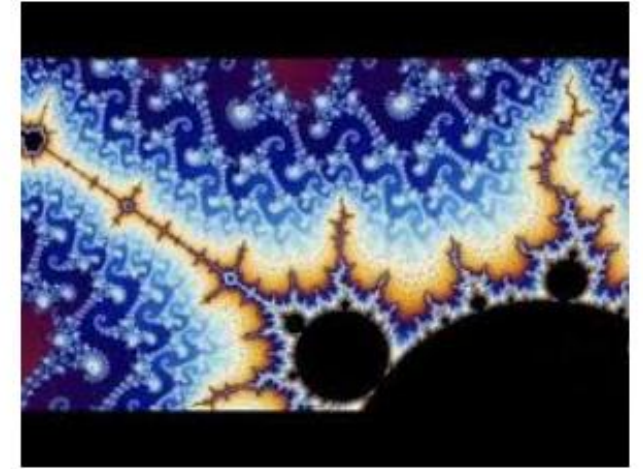
Really cool fractal patt...
pinterest.com



Fractal Never Ending Pattern Fractals Infinitely...
depositphotos.com



Explainer: what are fractals?
theconversation.com



Fractals The Hidden Dimension - YouTu...
youtube.com



An abstract computer ...
pinterest.com



Fractal Guide - Forging Mind
forgingmind.com



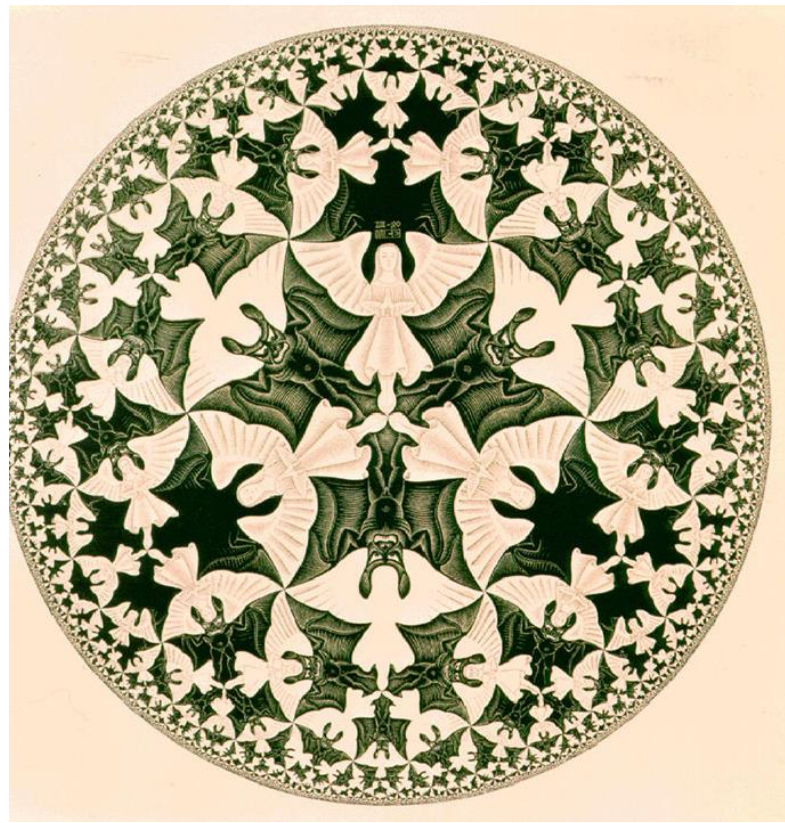
3d Abstract Computer Generated Fractal Desig...
123rf.com



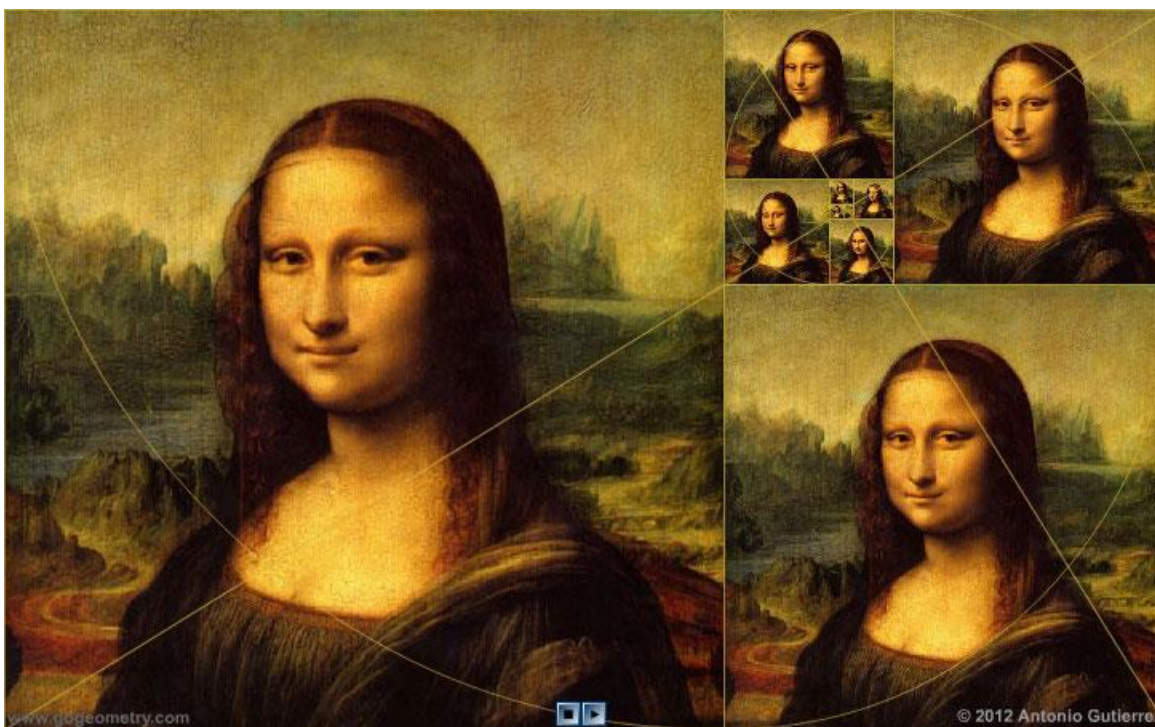
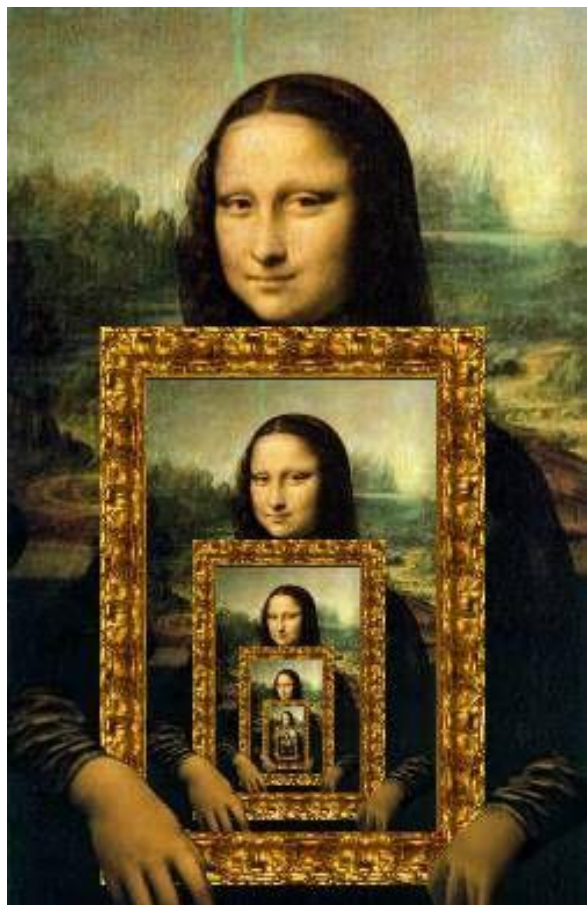
Mystic Universe, Fractals,...
fineartamerica.com



Fecursive graphics



Maurice Cornelius Escher





Iterations and recursion

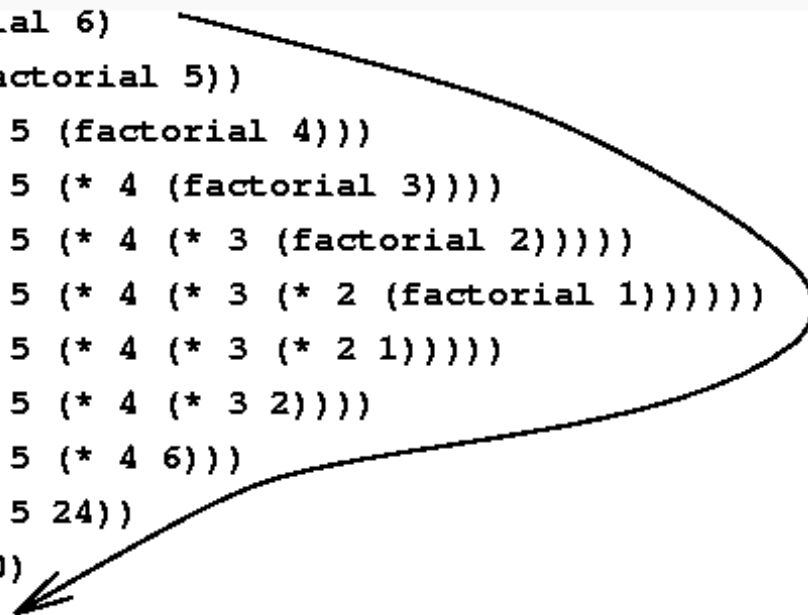
Iterations and recursion

The recursive process constantly says “I will remember this and then count it” at every step of the recursion. “Then” comes at the very end.

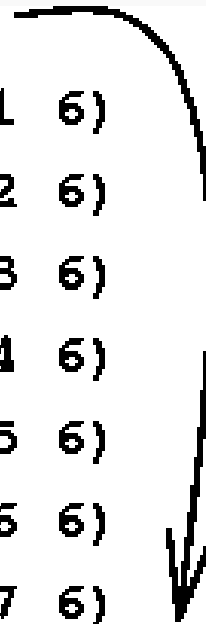
When a recursive process considers factorial 6, then it needs to remember 5 numbers in order to count them at the very end, when you can't get anywhere and you can no longer move downward recursively.

When we are in the next function call, these remembered numbers are stored somewhere outside in memory (in a zone named **stack**).


```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```



```
(factorial 6)
(fact-iter 1 1 6)
(fact-iter 1 2 6)
(fact-iter 2 3 6)
(fact-iter 6 4 6)
(fact-iter 24 5 6)
(fact-iter 120 6 6)
(fact-iter 720 7 6)
720
```



The scheme of recursive calculations



Recursion in programming

General structure of recursive function

```
if (elementary case)
    return <value for elementary case>
else return <value calculated by
            recurrent formula>
```

Simple example - power of 2 function:

```
long power2(int n)
{
    if ( n == 0)
        return 1; else return power2(n-1) * 2;
}
```

Convert base 10 integer to binary

- Base case. Return a value for small N . (0 or 1)
- Reduction step. Assuming that it works for smaller values of its argument, use the function to compute a return value for N.

```
1 // print binary representation of decimal n.
2 // Not create a binary copy in string or array
3
4 #include <stdio.h>
5
6 void tentobin(int n)
7 {
8     int k;
9     if (n < 2 ) printf("%d", n);
10    else { k = n % 2; tentobin(n / 2); printf("%d",k); }
11 }
12
13 int main()
14 {
15     int d;
16     scanf("%d", &d);
17     tentobin(d);
18 }
```


Proof that the program works correctly

1. The function. For $n = 1$ works correctly, return 1.
2. Suppose function works correctly for $n = z$.
3. Let $n = z + 1$. the function divide solution into 2 parts:
call of $n / 2$, which is less then z , so is correct and $n \% 2$ which is
0 or 1, less then z , also correct.

General method used to proof that the recursive program is correct –
mathematical induction.

Convert base 10 integer to base b ($b < 17$)

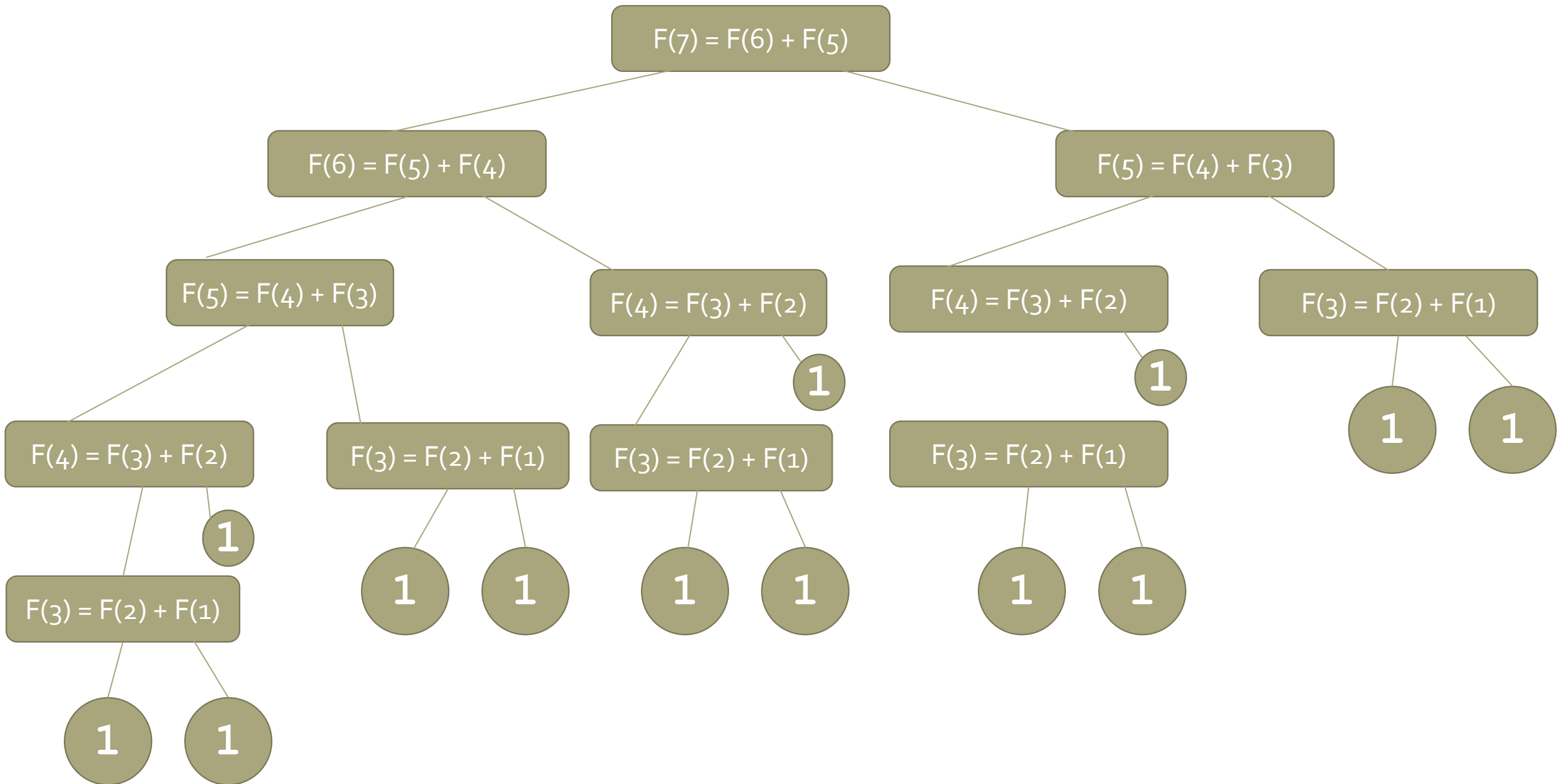
- Base case. Return a value for small N. (from 0 to $b - 1$)
- Reduction step. Assuming that it works for smaller values of its argument, use the function to compute a return value for N.

```
1  #include <stdio.h>
2
3  char z[100];
4  int strl;
5
6  void addstring(int n, int k)
7  {
8      if (n < 10) z[k] = 48 + n; else z[k] = 55 + n;
9  }
10 void tentob(int n, int b, int k)
11 {
12     if (n < b) { addstring(n, k); z[k + 1] = '\0'; strl = k + 1;}
13     else { tentob(n / b, b, k + 1); addstring(n % b, k); }
14 }
15 void inversestring(char *z, int left, int right)
16 {
17     char c = z[left];
18     if (left >= right) return;
19     else { z[left] = z[right]; z[right] = c;
20           inversestring(z, left + 1, right - 1);
21           return;
22     }
23 }
24 int main()
25 {
26     int d, b, l;
27     scanf("%d %d", &d, &b);
28     tentob(d, b, 0);           // convert to base b. in string Z.
29     inversestring(z, 0, strl - 1);
30     printf("%s\n", z);
31 }
```

«Bad»
recursion

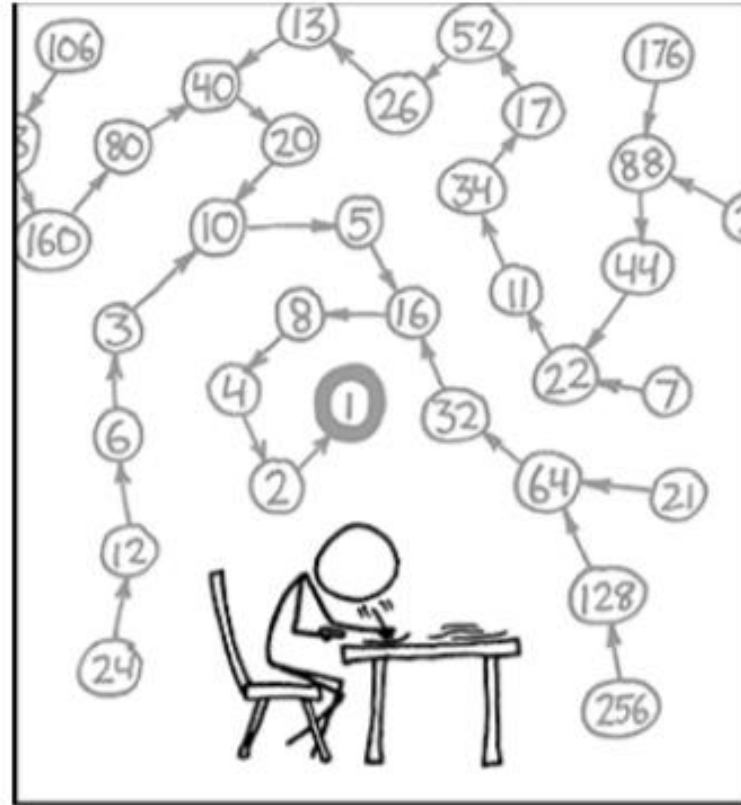
(Java)

```
// Warning: spectacularly inefficient.  
public static long fibonacci(int n) {  
    if (n == 0) return 0;  
    if (n == 1) return 1;  
    return fibonacci(n-1) + fibonacci(n-2);  
}
```



«Bad»
recursion 2

Collatz
sequence



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

Amazing fact.
No one knows
whether or not
this sequence
terminates for
all N (!)

```

1
2
3 #include <stdio.h>
4
5 void collatz(int n)
6 { printf("%d ", n);
7   if (n == 1) return;
8   if (n % 2 == 0) collatz(n / 2); else collatz(3 * n + 1);
9 }
10
11 int main()
12 {
13     int d;
14     scanf("%d", &d);
15     collatz(d);
16 }

```

C:\Users\CTI UST\Documents\codeblocks\TUM\Recursion\collatz.exe

106

106 53 160 80 40 20 10 5 16 8 4 2 1

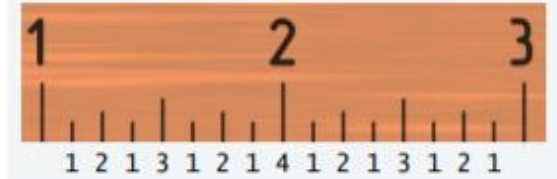
Process exited after 2.962 seconds with return value 0

Press any key to continue . . .

Ruller, how it works

ruller(n): create subdivisions of a ruler to $1/2^n$ inches.

- Return one space for $n = 0$.
- Otherwise, sandwich n between two copies of ruller($n-1$).



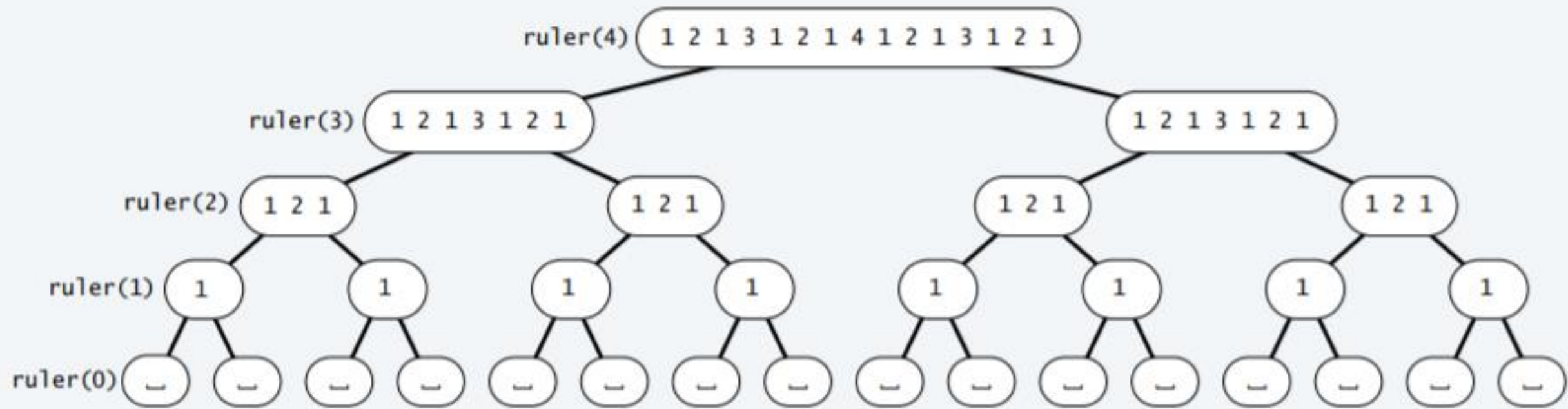
```
1  #include <stdio.h>
2
3  void ruller(int n)
4  { if (n == 0) { printf(" ");
5    return; }
6    else {
7        ruller(n - 1);
8        printf("%d", n);
9        ruller(n - 1);
10    }
11 }
12 int main()
13 {
14     int d;
15     scanf("%d", &d);
16     ruller(d);
17 }
```

C:\Users\CTI UST\Documents\codeblocks\TUM\Recursion\ruller.exe

```
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
-----
Process exited after 2.538 seconds with return value 0
Press any key to continue . . .
```

Use a *recursive call tree*

- One node for each recursive call.
- Label node with return value after children are labeled.





Towers of Hanoi

Understanding

A legend of uncertain origin

- $n = 64$ discs of differing size; 3 posts; discs on one of the posts from largest to smallest.
- An ancient prophecy has commanded monks to move the discs to another post.
- When the task is completed, *the world will end*.

Rules

- Move discs one at a time.
- Never put a larger disc on a smaller disc.

Q. Generate list of instruction for monks ?

Q. When might the world end ?

$n = 10$

before



after



Recursive solving

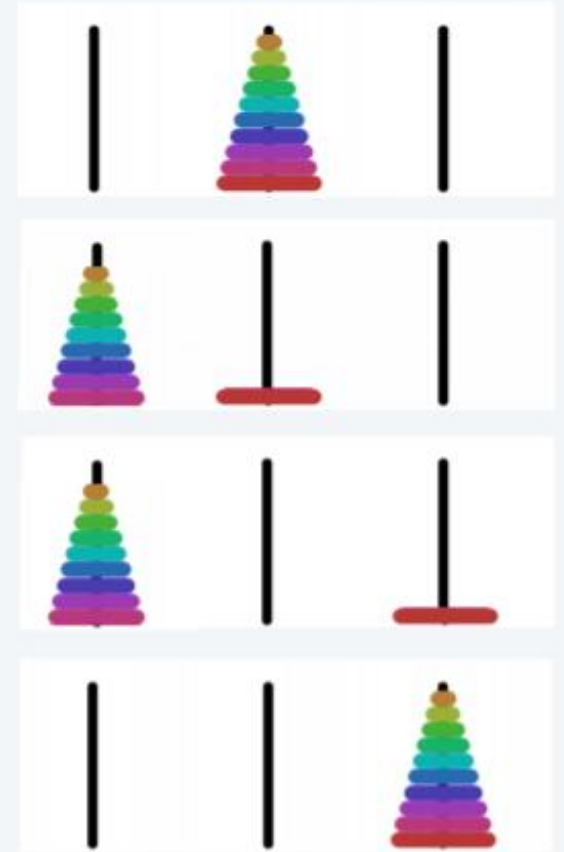
For simple instructions, use cyclic wraparound

- Move *right* means 1 to 2, 2 to 3, or 3 to 1.
- Move *left* means 1 to 3, 3 to 2, or 2 to 1.



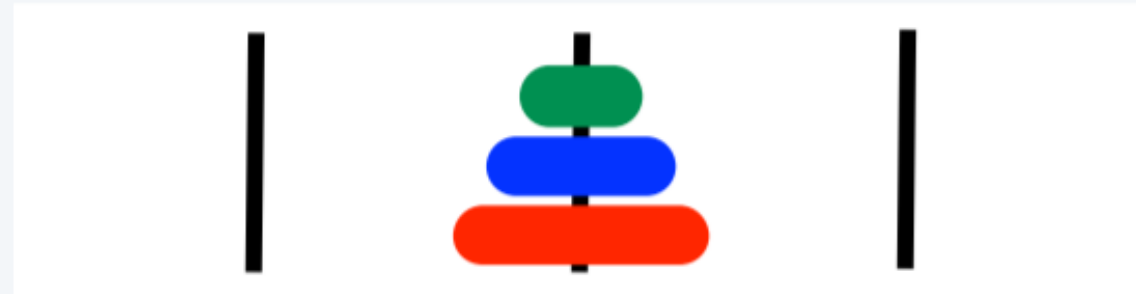
A recursive solution

- Move $n - 1$ discs to the left (recursively).
- Move largest disc to the *right*.
- Move $n - 1$ discs to the left (recursively).



Simple
solution

($n = 3$)



1R

2L

1R

3R

1R

2L

1R



1R



2L



1R



3R



1R



2L



1R



Towers of Hanoi

FaQ

Q. Generate list of instructions for monks ?

A. (Long form). 1L 2R 1L 3L 1L 2R 1L 4R 1L 2R 1L 3L 1L 2R 1L 5L 1L 2R 1L 3L 1L 2R 1L 4R ...

A. (Short form). Alternate "1L" with the only legal move not involving the disc 1.

"L" or "R" depends on whether n is odd or even

Q. When might the world end ?

A. Not soon: need $2^{64} - 1$ moves.

Note: Recursive solution has been proven optimal.

<i>moves per second</i>	<i>end of world</i>
1	5.84 billion centuries
1 billion	5.84 centuries



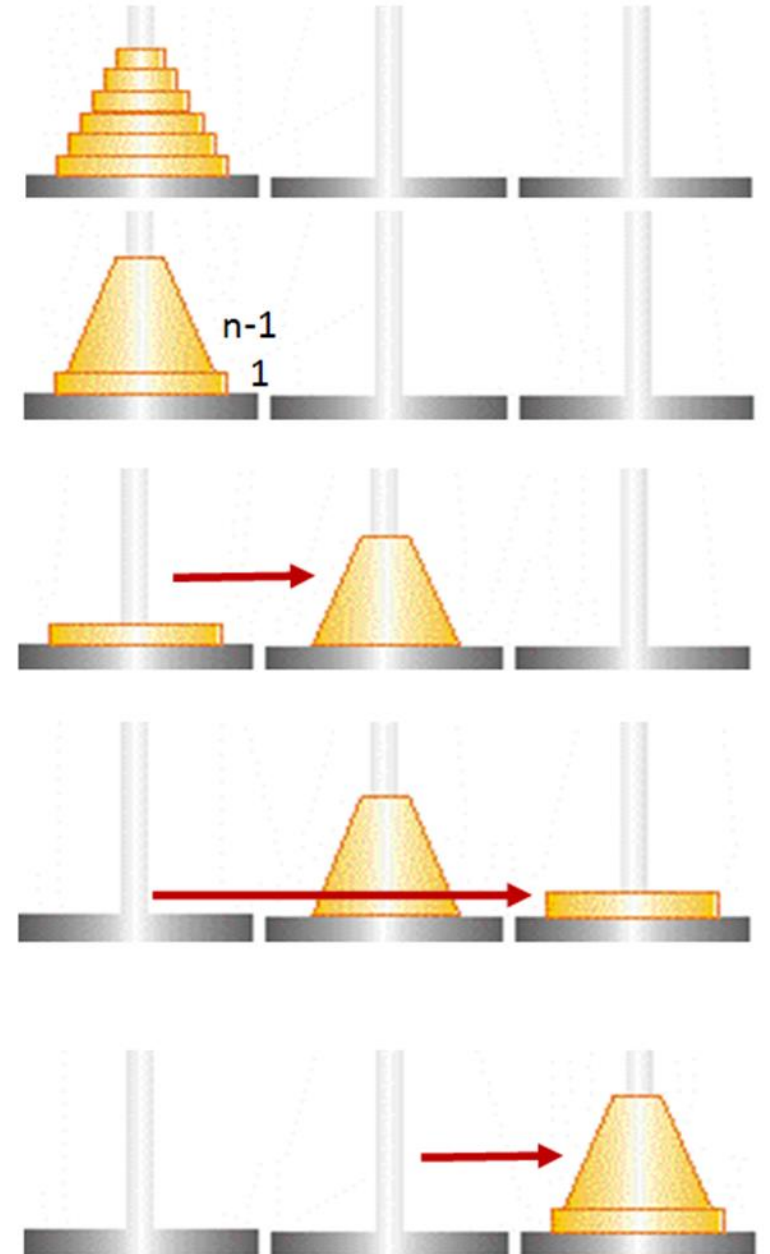
Algoritm:

Pas 0. Primele $n-1$ discuri se consideră ca un “megadisc”, care poate fi transferat direct prin o “megamutare” (aceeași problemă de dimensiune $n-1$)

Pas 1. Prin o “megamutare” se deplasează $n-1$ discuri de pe tija A pe tija B. ($\text{hanoi}(A,B,C,n-1)$)

Pas 2. Prin o mutare simplă se deplasează 1 disc de pe tija A pe tija C.

Pas 3. Prin o “megamutare” se deplasează $n-1$ discuri de pe tija B pe tija C. ($\text{hanoi}(B,C,A,n-1)$)



The code

solution in text
mode

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int *a, *b, *c, m, i;
5
6  void print(int n)
7  { int j;
8    for (j = 0; j < n; j++)
9        printf("\n %3d %3d %3d ", a[j], b[j], c[j]);
10    printf("\n-----\n") ;
11 }
```

Recursive model

```
12 void muta (int *x, int *y)
13 { int k = 0, d;
14   while (x[k] == 0) k++;
15   d = x[k]; x[k] = 0; //discul din pozitia k este luat de pe X
16   k = m - 1;
17   while (y[k] != 0) k--;
18   y[k]=d;           // si pus in ultima pozitie libera pe Y
19 }
20 void hanoi(int *a, int *b, int *c, int n)
21 { if (n==1)
22     { muta(a, b); print (m);}
23   else
24     { hanoi (a, c, b, n - 1);
25       muta (a, b);
26       print(m);
27       hanoi(c,b,a, n - 1);
28     }
29 }
```

Main function

```
31 int main()  
32 { scanf("%d", &m);  
33   a = (int*) calloc(m, sizeof(int));  
34   b = (int*) calloc(m, sizeof(int));  
35   c = (int*) calloc(m, sizeof(int));  
36   for (i = 0; i < m ; i++) a[i] = i + 1;  
37   print(m);  
38   hanoi(a, c, b, m);  
39 }
```

3

1	0	0
2	0	0
3	0	0

1

0	0	0
2	0	0
3	0	1

2

0	0	0
0	0	0
3	2	1

3

0	0	0
0	1	0
3	2	0

4

0	0	0
0	1	0
0	2	3

5

0	0	0
0	0	0
1	2	3

6

0	0	0
0	0	2
1	0	3

7

0	0	1
0	0	2
0	0	3

8

Homework

Blitz 2 started!

Write recursive functions to calculate:

1. $n!$ ($n < 20$)

2. a^b (a – real, b -integer) / in limits of **long**

3. “Bad” Fibonacci (N) $F(N) = F(N - 1) + F(N - 2)$

$$4. \quad C_n^k = C_n^{k-1} \times \frac{n-k+1}{k}$$

5. Greatest common divider (Euclide algorithm, recursive!)

6. Recursive determination of the validity of an arithmetic expression, for which it is known that :

`<digit> ::= 0|1|2|3|4|5|6|7|8|9|0`

`<number> ::= <digit> {<digit>}`

`<sign> ::= +|-`

`<expression> ::= <number> {<sign> <number>}`

No words...
only recursion
:)

