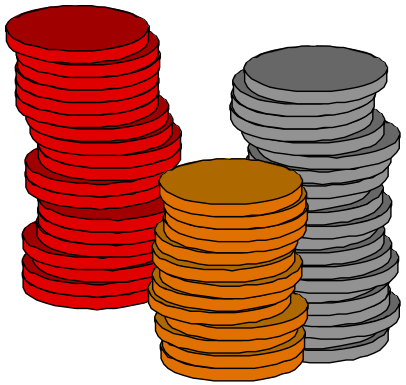
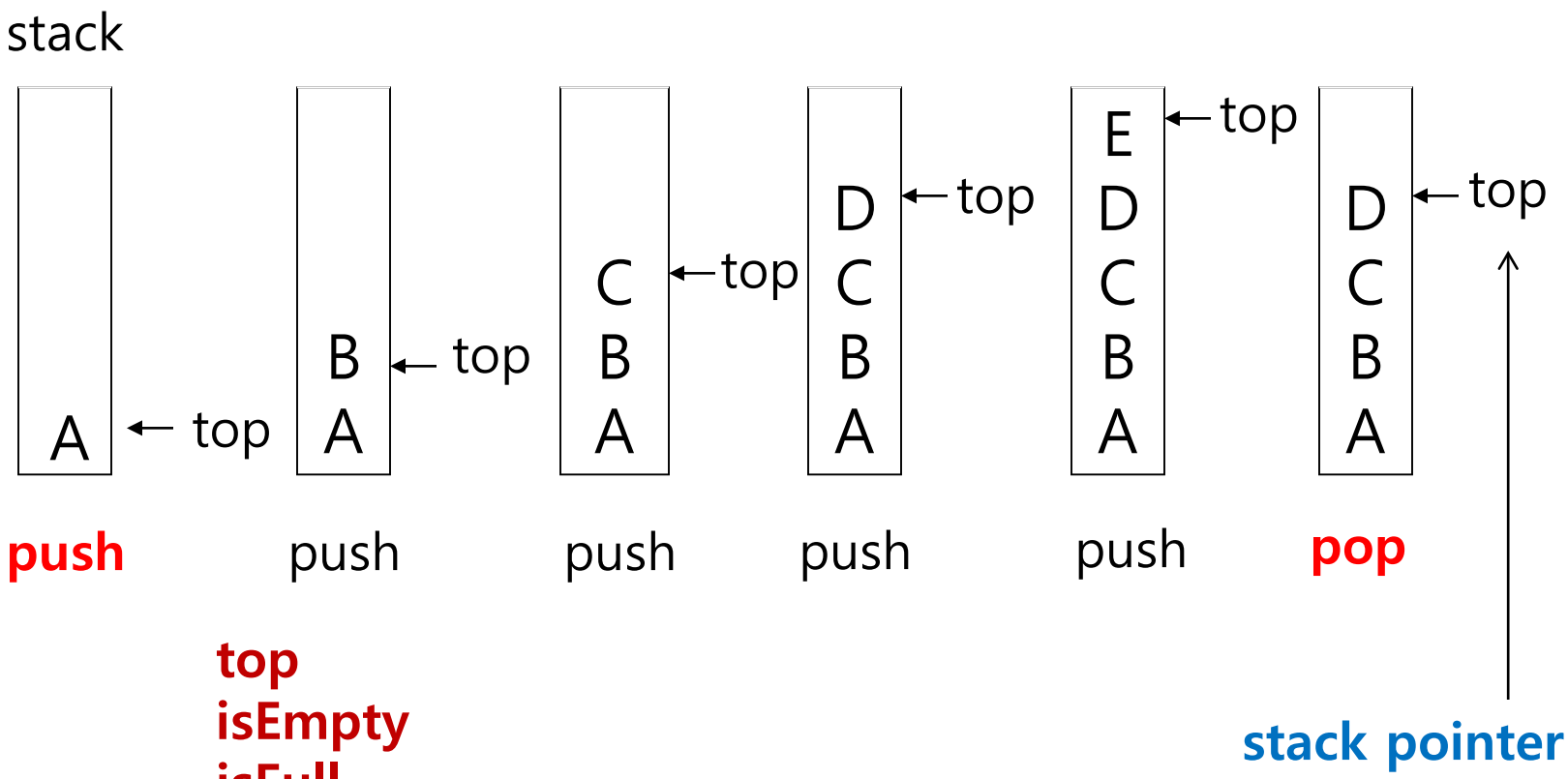
What is a stack?

- 액세스 원리: **LAST IN FIRST OUT (후입선출)**
  - = **LIFO**
  - **Single pointer로 관리 (stack pointer)**
- Example
  - 책 쌓기, 접시 쌓기, .. 등 대부분의 쌓기는 stack의 구조로 동작한다



Last In First Out



Stack Applications

- Real life
  - Pile of books
  - Plate trays
- More applications related to computer science
  - **Program execution stack**
  - Evaluating expressions

Function Call and Return To …

```
public void a()
{ ...; b(); ...}
public void b()
{ ...; c(); ...}
public void c()
{ ...; d(); ...}
public void d()
{ ...; e(); ...}
public void e()
{ ...; c(); ...}
```

return address in d()

return address in c()

return address in e()

return address in d()

return address in c()

return address in b()

return address in a()

Stack ADT

objects: a **finite ordered list** with zero or more elements.
methods:

for all stack ∈ Stack, item ∈ element, max_stack_size ∈ positive integer
Stack createS(max_stack_size) ::=
create an empty stack whose maximum size is
max_stack_size
Boolean isFull(stack, max_stack_size) ::=
if (number of elements in stack == max_stack_size)
return TRUE
else return FALSE
Stack push(stack, item) ::=
if (IsFull(stack)) stack_full
else insert item into top of stack and **return**
Boolean isEmpty(stack) ::=
if(stack == CreateS(max_stack_size))
return TRUE
else return FALSE
Element pop(stack) ::=
if(IsEmpty(stack)) **return**
else remove and return the item on the top of the stack.

Array-based Stack Implementation

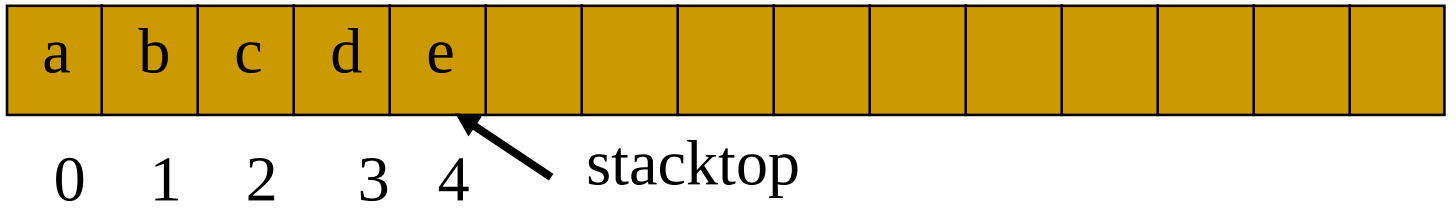
- Allocate an array of some size (pre-defined)
  - Maximum N elements in stack
- Bottom stack element stored at element 0
- last index in the array is the *top*
- **Increment *top* when one element is pushed, decrement after pop**

Stack Implementation: CreateS, isEmpty, isFull

- Stack의 생성 : (기본 변수 혹은 구조체의) 1차원 배열, 혹은 Linked list

```
#define MAX_STACK_SIZE      100     /* maximum stack size */
int stack[MAX_STACK_SIZE] ;          /* integer stack */

typedef struct {                     /* 구조체 배열로 생성한 스택 */
        int key;
        /* other fields */
        } STACK ;
STACK stack[MAX_STACK_SIZE];

int stacktop = -1;
```

Push




```
void push(element item)
{
 /* add an item to the global stack */
    if (stacktop >= MAX_STACK_SIZE-1)  {
         stack_full( );
         return;
    }
    stack[++stacktop] = item;
}
```

StackFull()

```
stack_full()
{
    printf("Stack is full, cannot add element.");
    exit(EXIT_FAILURE);
    // 프로그래머의 의도에 따라 여러가지 방법으로 구현 가능하다
}
```

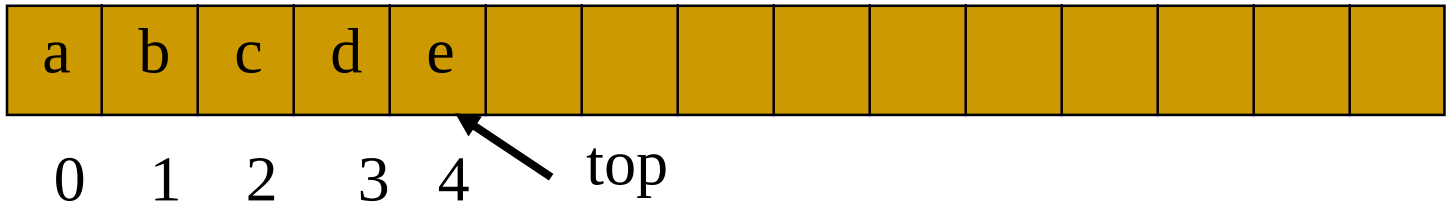
StackFull()/Dynamic Array

- Use a variable called *capacity* in place of MAX_STACK_SIZE
- Initialize this variable to (say) 1
- When stack is full, double the capacity using REALLOC
- This is called *array doubling*

StackFull()/Dynamic Array

```
int capacitiy = 1;
.
.
.
stack_full()
{
      realloc(stack, 2*capacity*sizeof(stack);
      capacity *= 2;          // 혹은 capacity++ ;
}
```

Pop



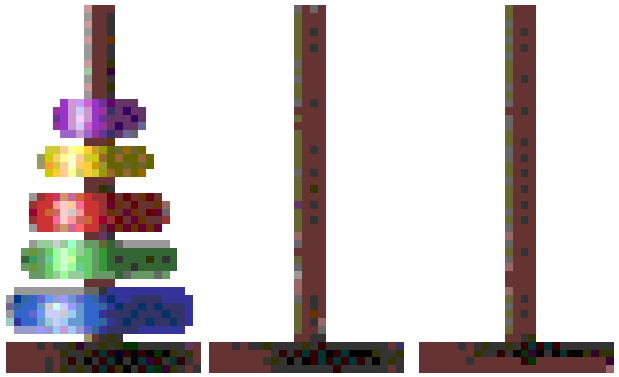

```
element_type pop()
{
 /* return the top element from the stack */
    if (top == -1)
        return stack_empty( );              /* returns and error key */
    return stack[stacktop--];
 }
```

```
#include <stdio.h>
#define MAX_STACK_SIZE 100
int stack[MAX_STACK_SIZE];
int stacktop = -1;

void push(int item) {
if(stacktop >= MAX_STACK_SIZE - 1) {
        printf("stack_full()\n");
        return;
}
stack[++top] = item;
}

int pop() {
if(stacktop == -1) {
        printf("stack_empty()\n"); exit();
}
return stack[stacktop--];
}
```

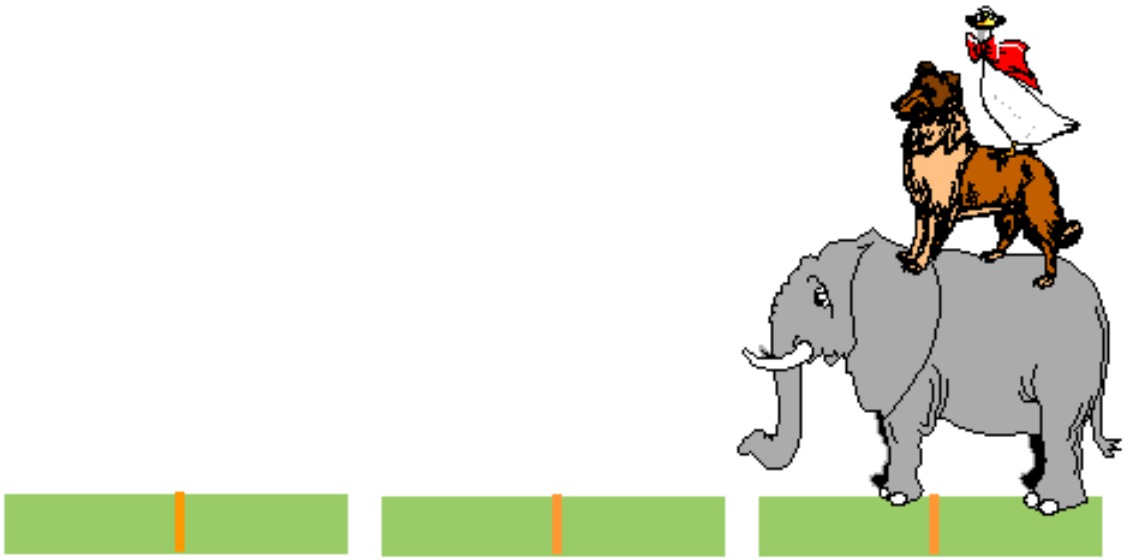
```
int isempty() {
{ if( stacktop == -1 ) return(1); else return(0); }
int isfull()
{ if ( stacktop >= MAX_STACK_SIZE -1 ) return(1);
else return(0); }

int main()
{
        int e;
        push(20);
        push(40);
        push(60);
        printf(" Begin Stack Test ...\n");
        while(!isempty()) {
                e = pop();
                printf("value = %d\n", e);
        }
}
```

The Towers of Hanoi
A Stack-based Application

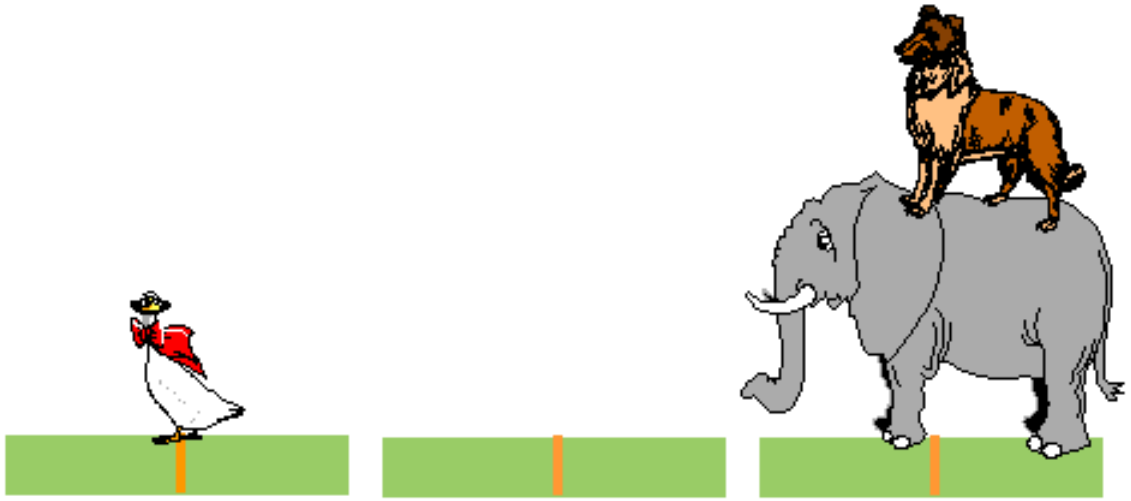
- GIVEN: three poles
- a set of discs on the first pole, discs of different sizes, the smallest discs at the top
- GOAL: move all the discs from the left pole to the right one.
- CONDITIONS: only one disc may be moved at a time.
- A disc can be placed either on an empty pole or on top of a larger disc.



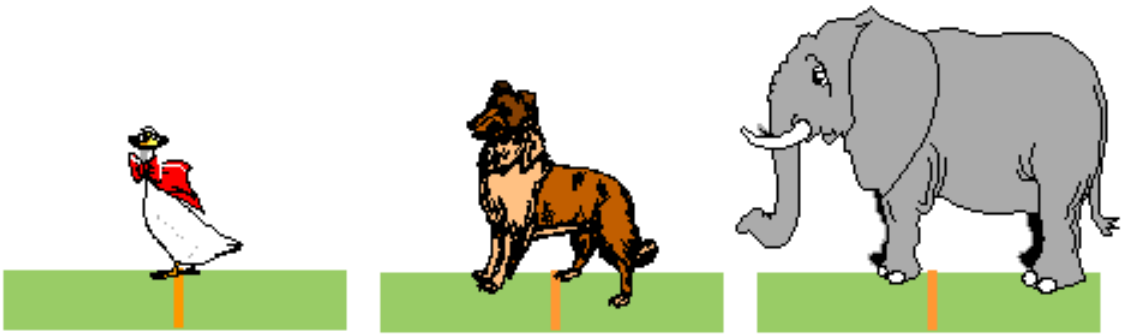
Towers of Hanoi



Towers of Hanoi



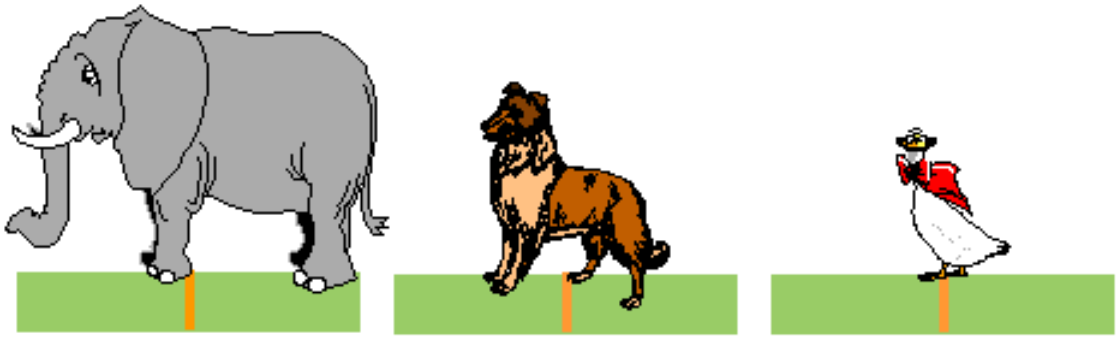
Towers of Hanoi



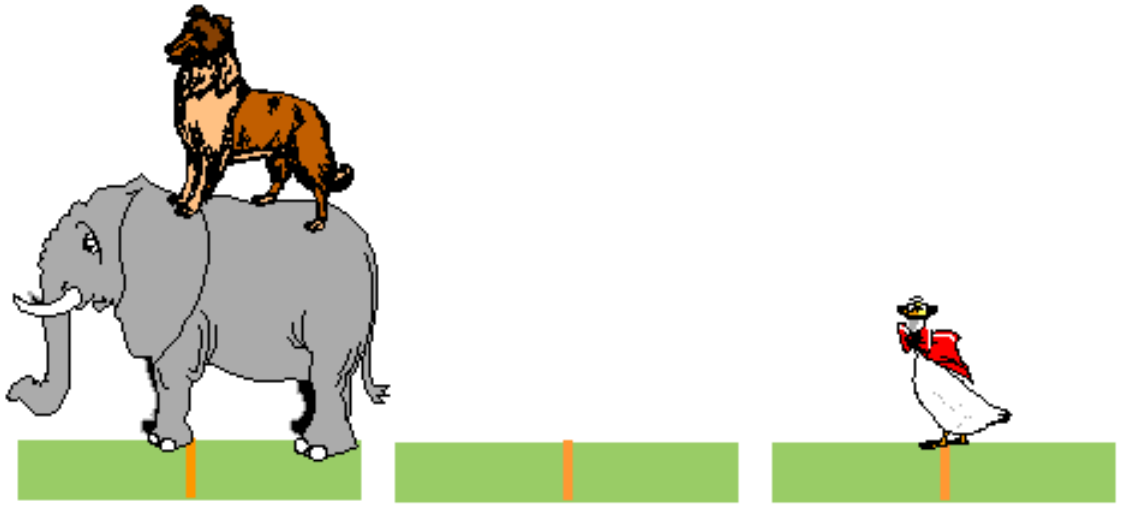
Towers of Hanoi

Towers of Hanoi

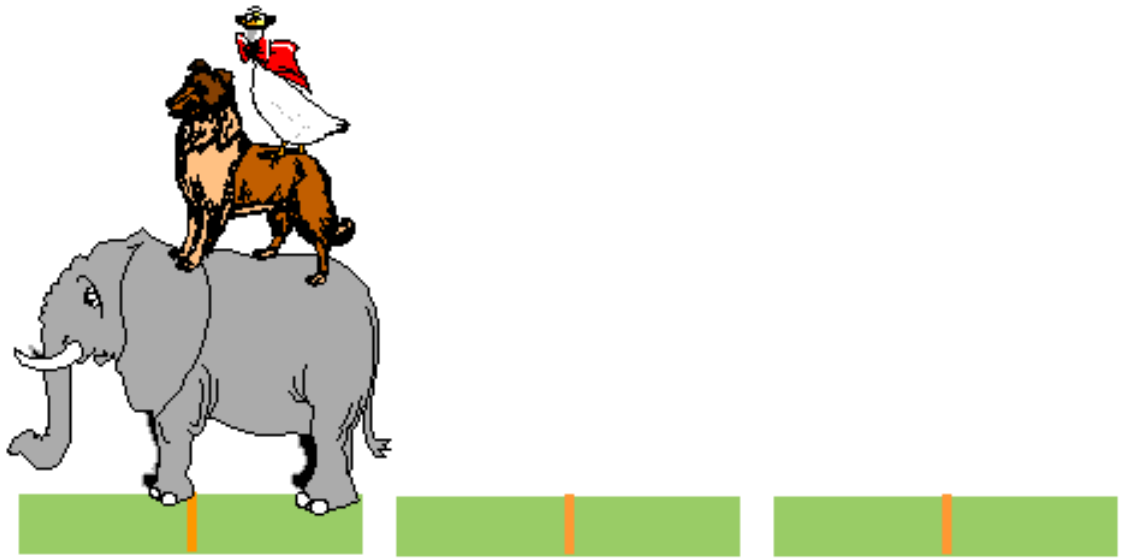
Towers of Hanoi



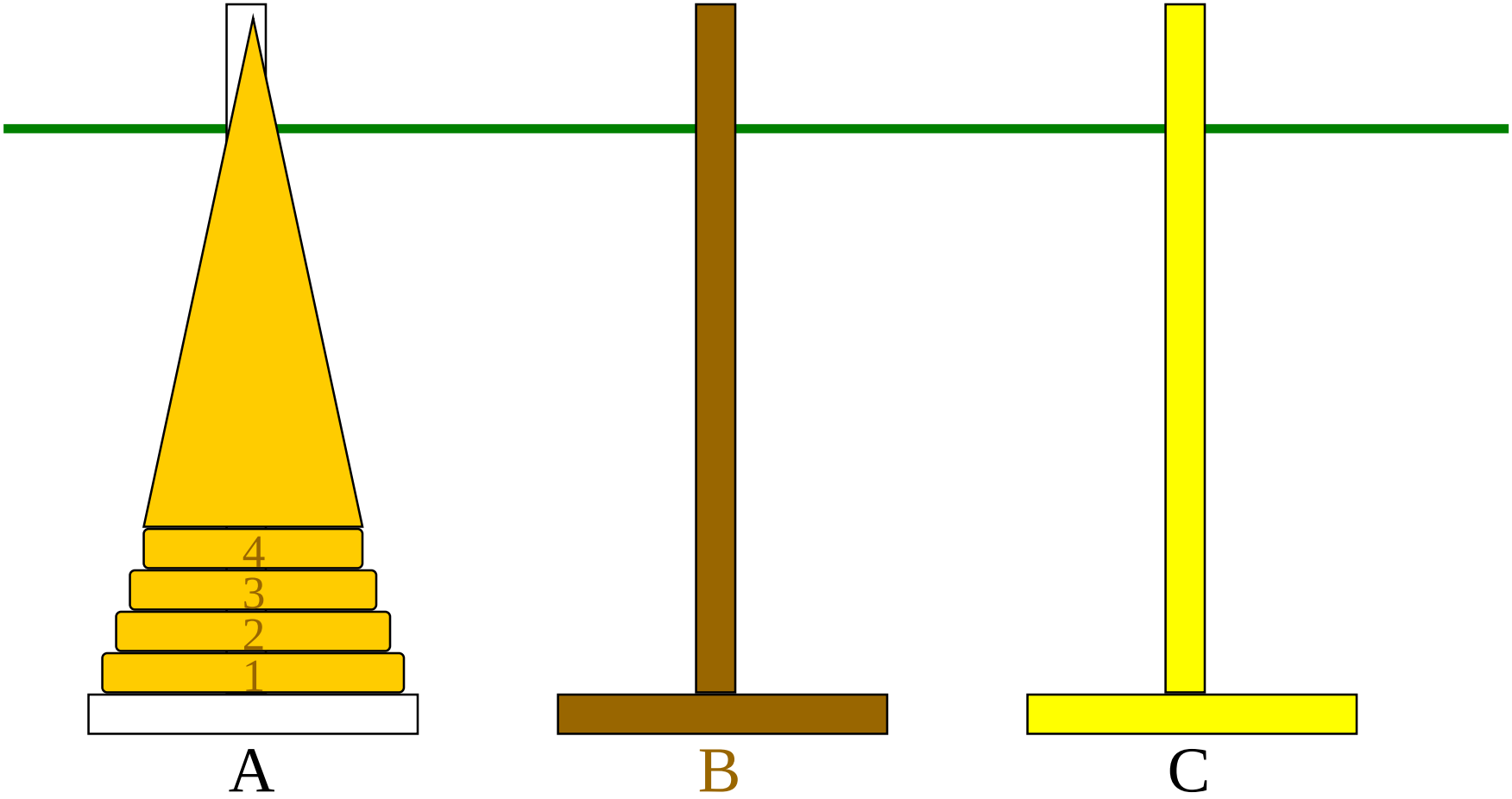
Towers of Hanoi



Towers of Hanoi

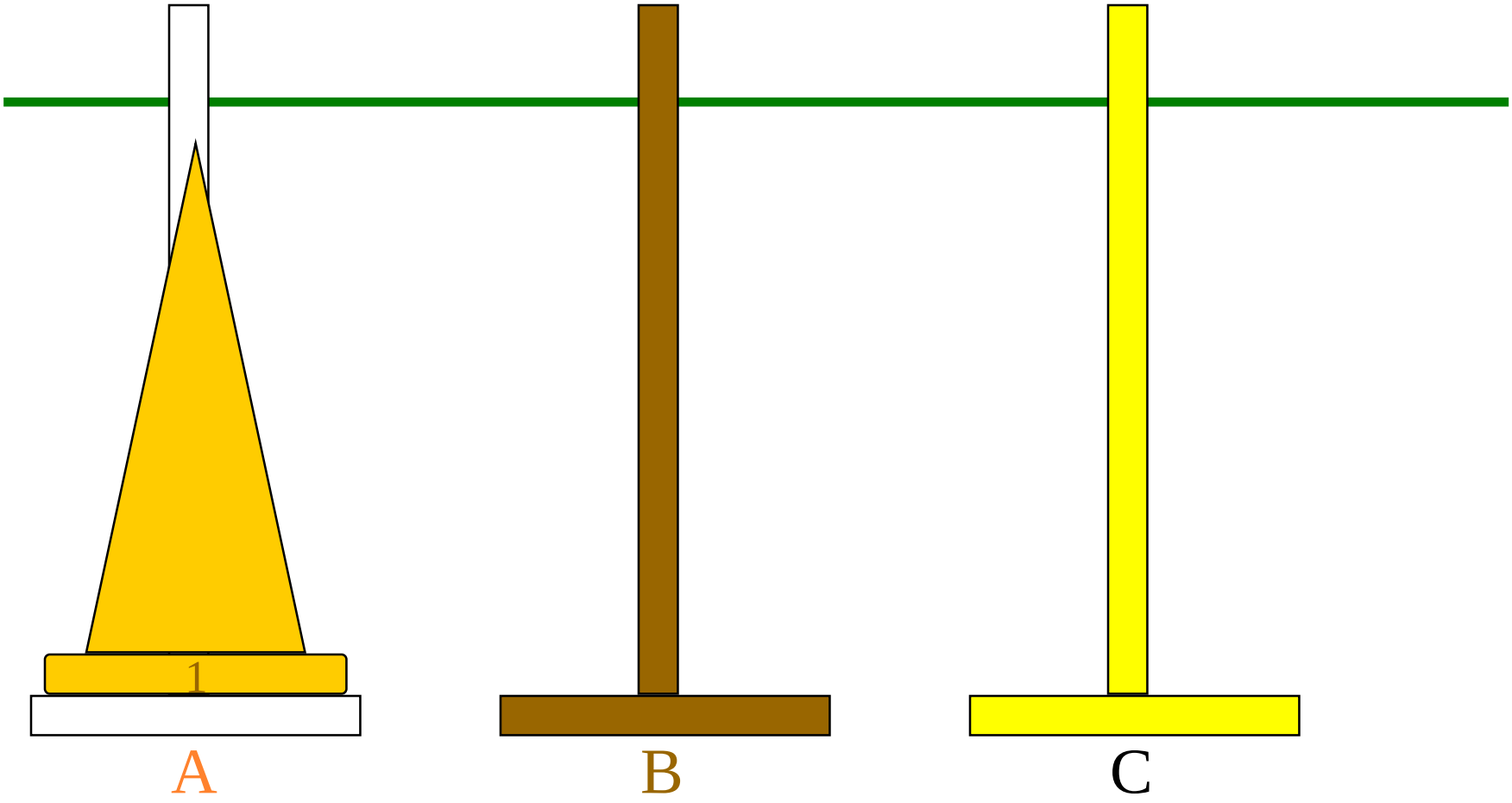


Towers Of Hanoi/Brahma



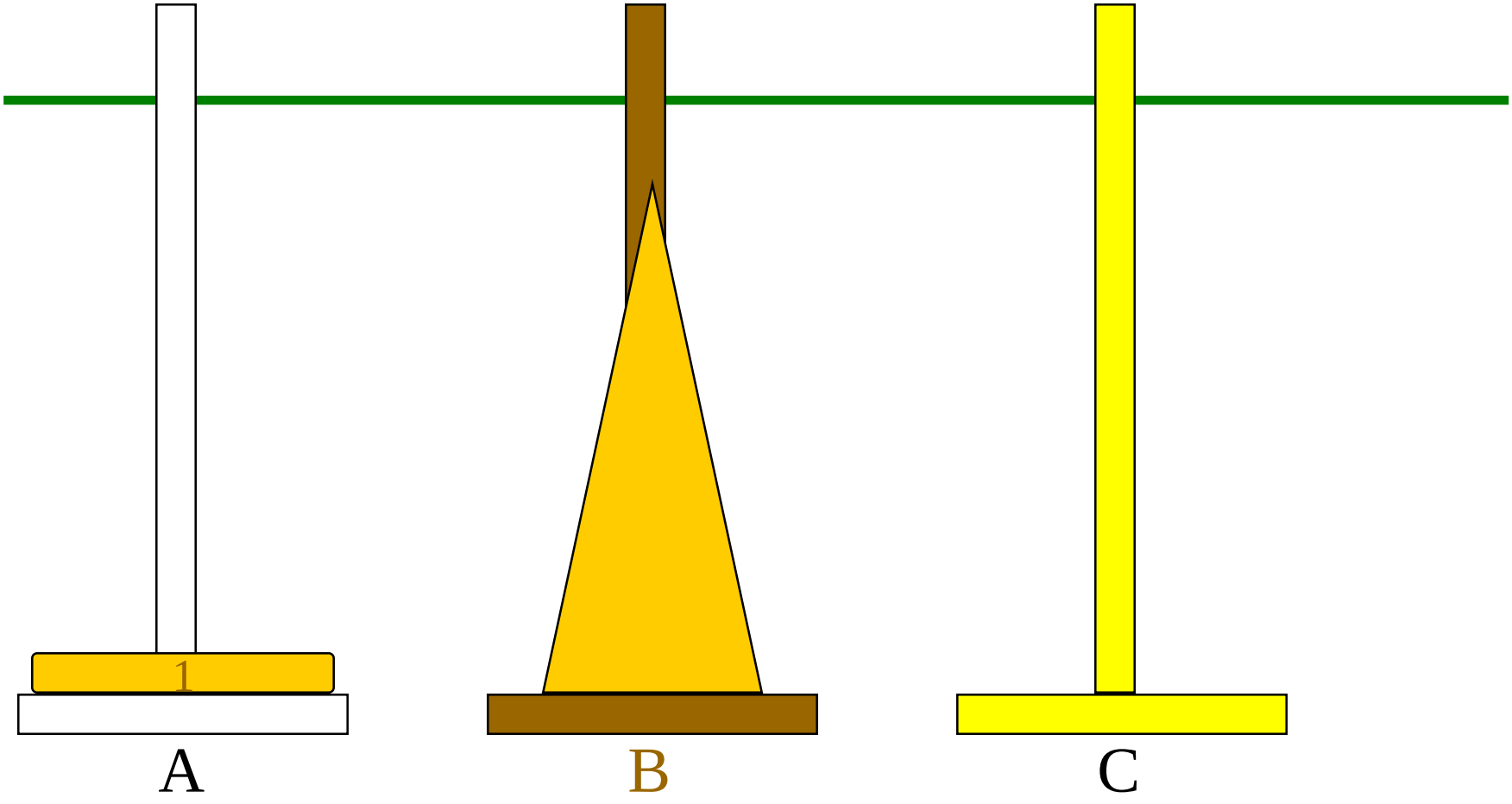
- 64 gold disks to be moved from tower A to tower C
- each tower operates as a stack
- cannot place big disk on top of a smaller one

Recursive Solution



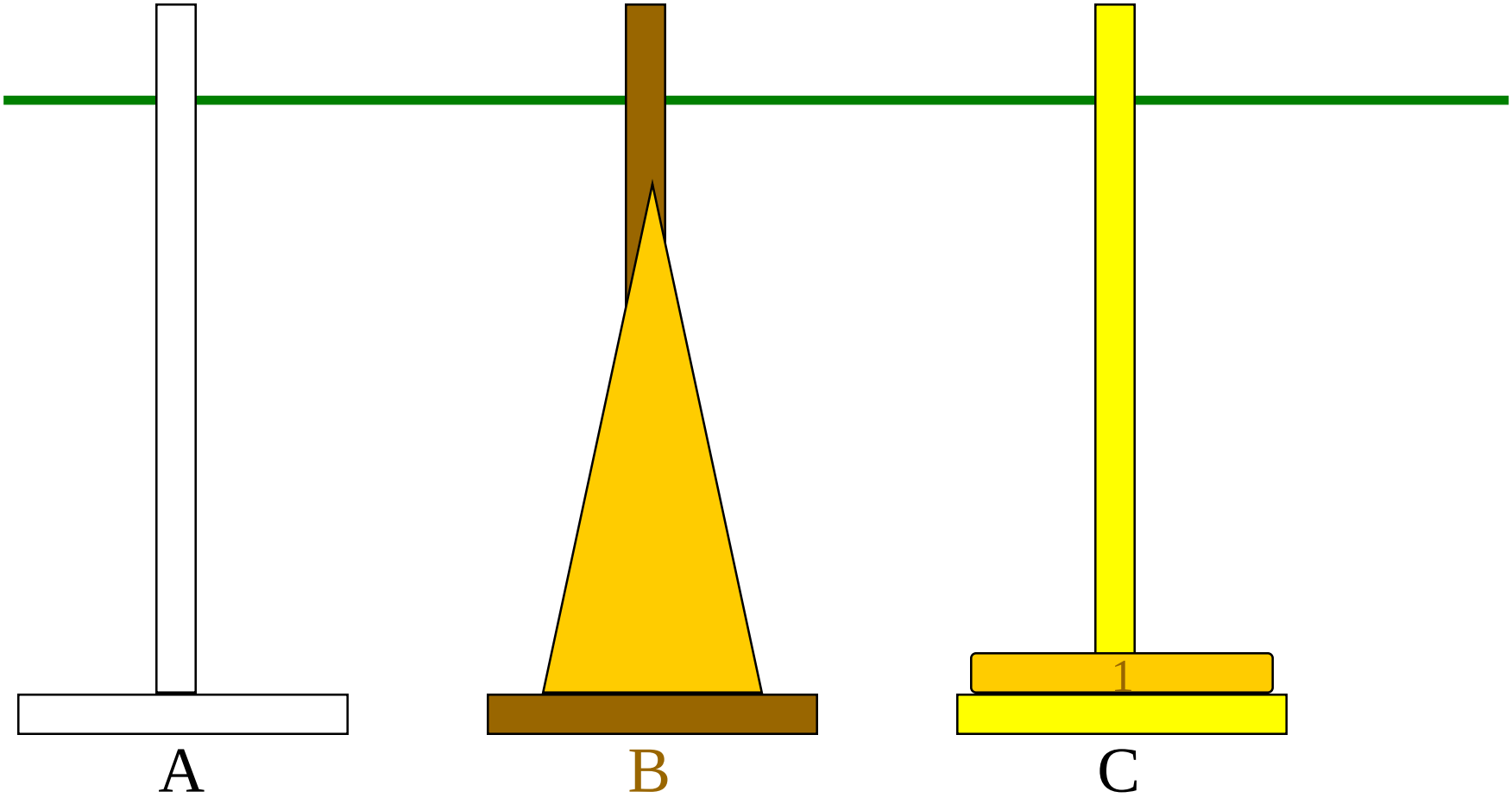
- n > 0 gold disks to be moved from A to C using B
- move top n-1 disks from A to B using C

Recursive Solution



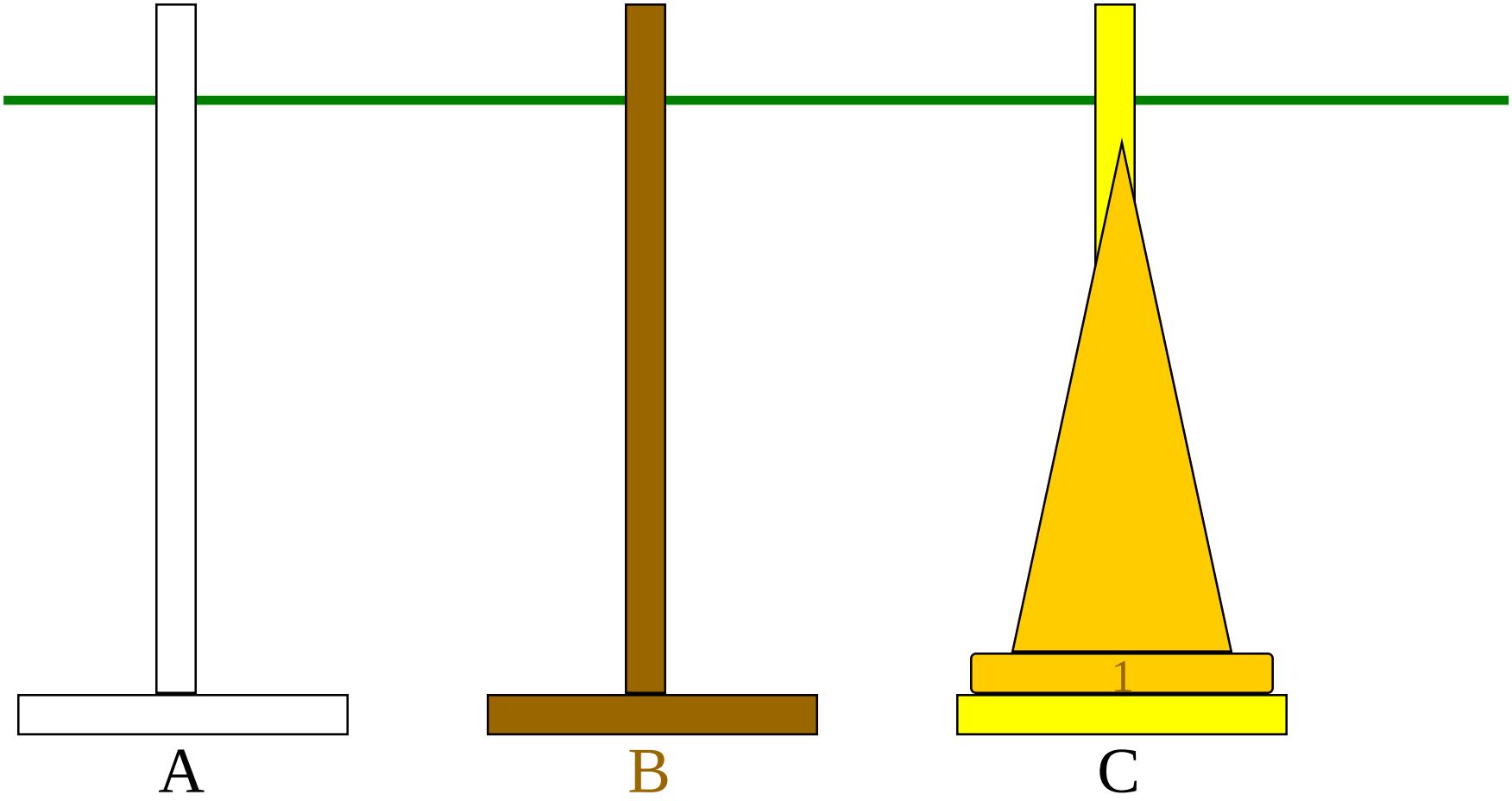
- move top disk from A to C

Recursive Solution



- move top n-1 disks from B to C using A

Recursive Solution



- moves(n) = 0 when n = 0
- moves(n) = 2*moves(n-1) + 1 = 2^n-1 when n > 0

Towers of Hanoi – Recursive solution

```
void hanoi (int discs,  Stack fromPole,  Stack toPole,  Stack aux) {
   disc d;
   if( discs >= 1) {
        hanoi(discs-1, fromPole, aux, toPole);
        d = fromPole.pop();
        toPole.push(d);
        hanoi(discs-1,aux, toPole, fromPole);
 }
```

Is the End of the World Approaching?

- Problem complexity 2^n
- 64 gold discs
- moves(64) = $1.8 * 10^{19}$ (approximately)
- Given 1 move a second

→ 600,000,000,000 years until the end of the world ☺