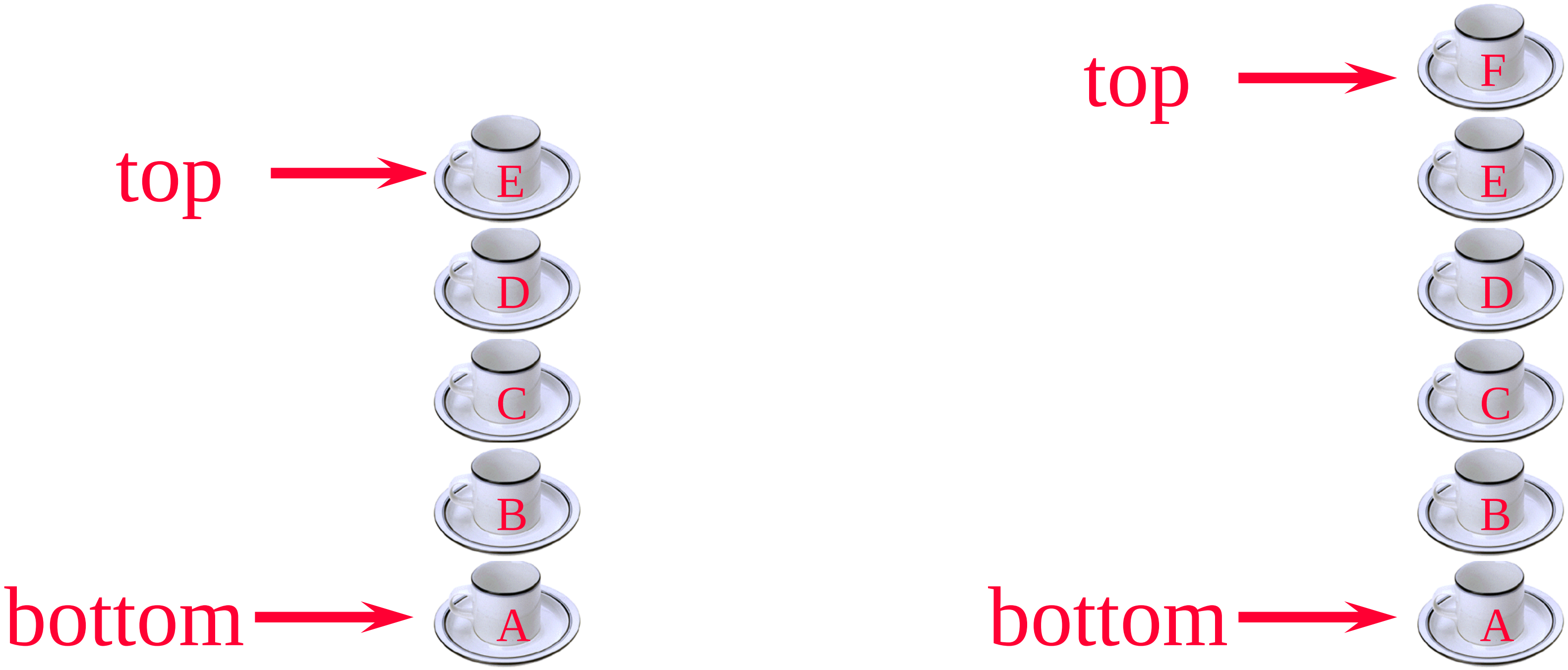
# Stacks

- Linear list.
- One end is called top.
- Other end is called bottom.
- Additions and removals from the top end only.

# Stack Of Cups



- Add a cup to the stack.
- Remove a cup from new stack.
- A stack is a LIFO list.

# The Interface Stack

```
public interface Stack
{
    public boolean empty();
    public Object peek();
    public void push(Object theObject);
    public Object pop();
}
```

# Reversing data items

- Reversing data items requires that a given set of data items be reordered so that the first and last items are exchanged, with all of the positions between the first and last also being relatively exchanged.

- For example, the list (2, 4, 7, 1, 6, 8) becomes
  (8, 6, 1, 7, 4, 2).

# Parentheses Matching

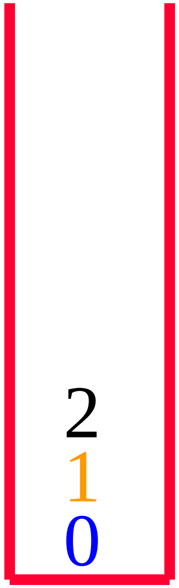
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)
  - Output pairs (u,v) such that the left parenthesis at position u is matched with the right parenthesis at v.
    - (2,6) (1,13) (15,19) (21,25) (27,31) (0,32) (34,38)
- (a+b))*((c+d)
  - (0,4)
  - right parenthesis at 5 has no matching left parenthesis
  - (8,12)
  - left parenthesis at 7 has no matching right parenthesis

# Parentheses Matching

- scan expression from left to right
- when a left parenthesis is encountered, add its position to the stack
- when a right parenthesis is encountered, remove matching position from stack

# Example

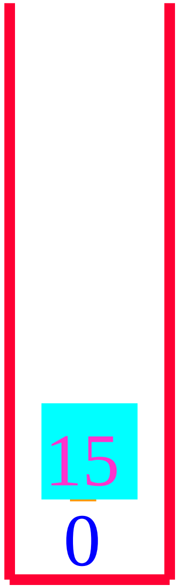
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



2
1
0

# Example

- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



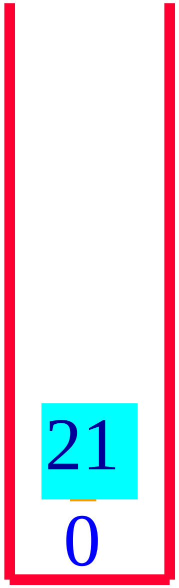
15

0

(2,6) (1,13)

# Example

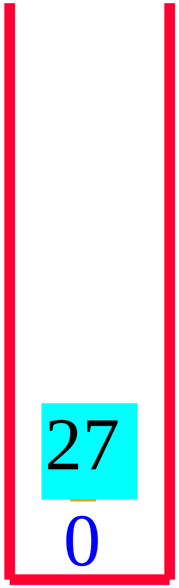
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19)

# Example

- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



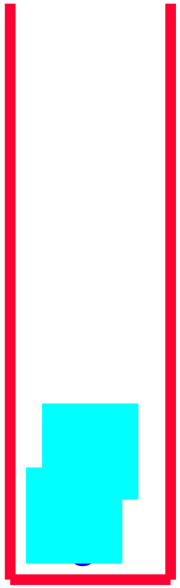
27

0

(2,6) (1,13) (15,19) (21,25)

# Example

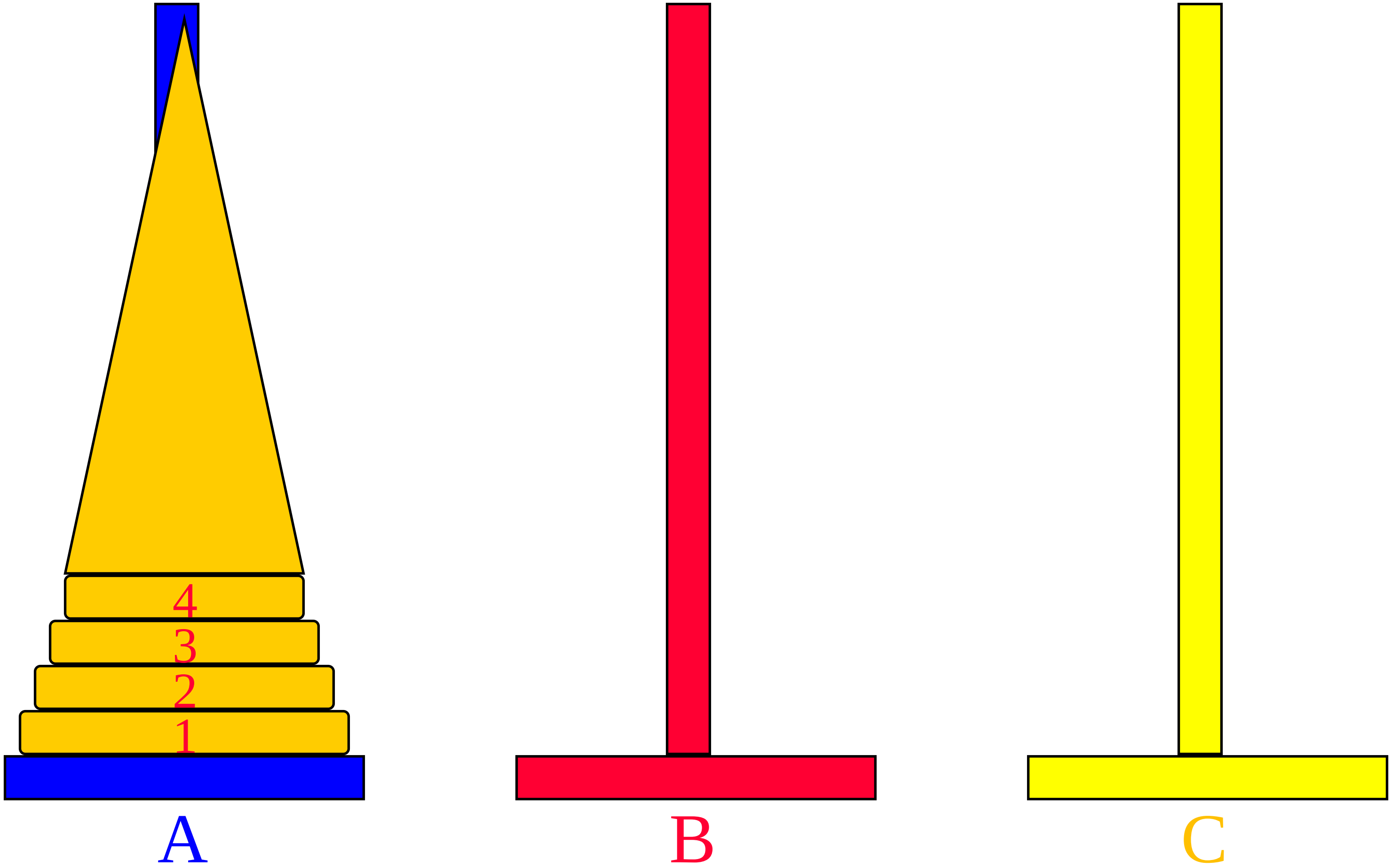
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19) (21,25)(27,31) (0,32)

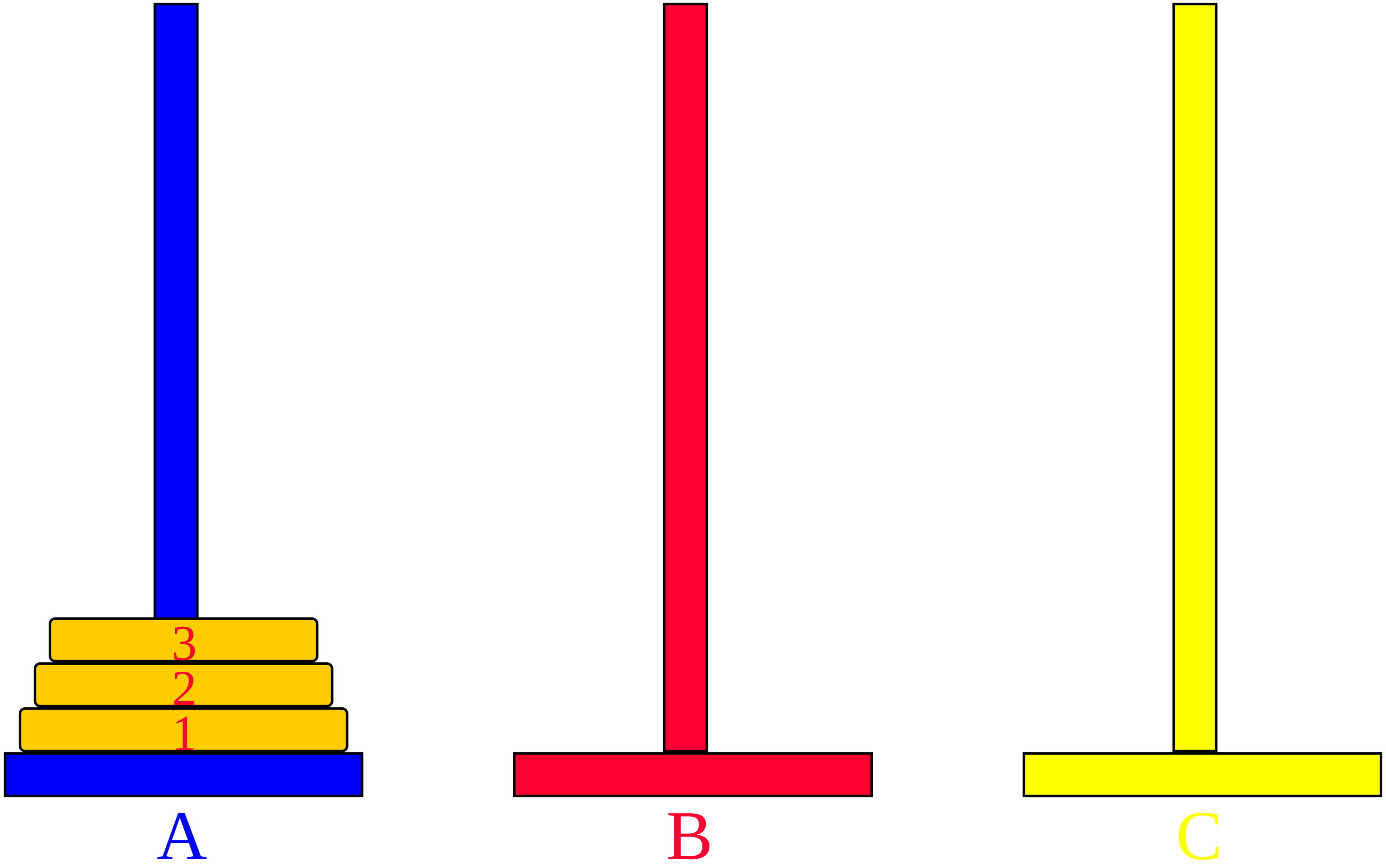
- and so on

# Towers Of Hanoi/Brahma



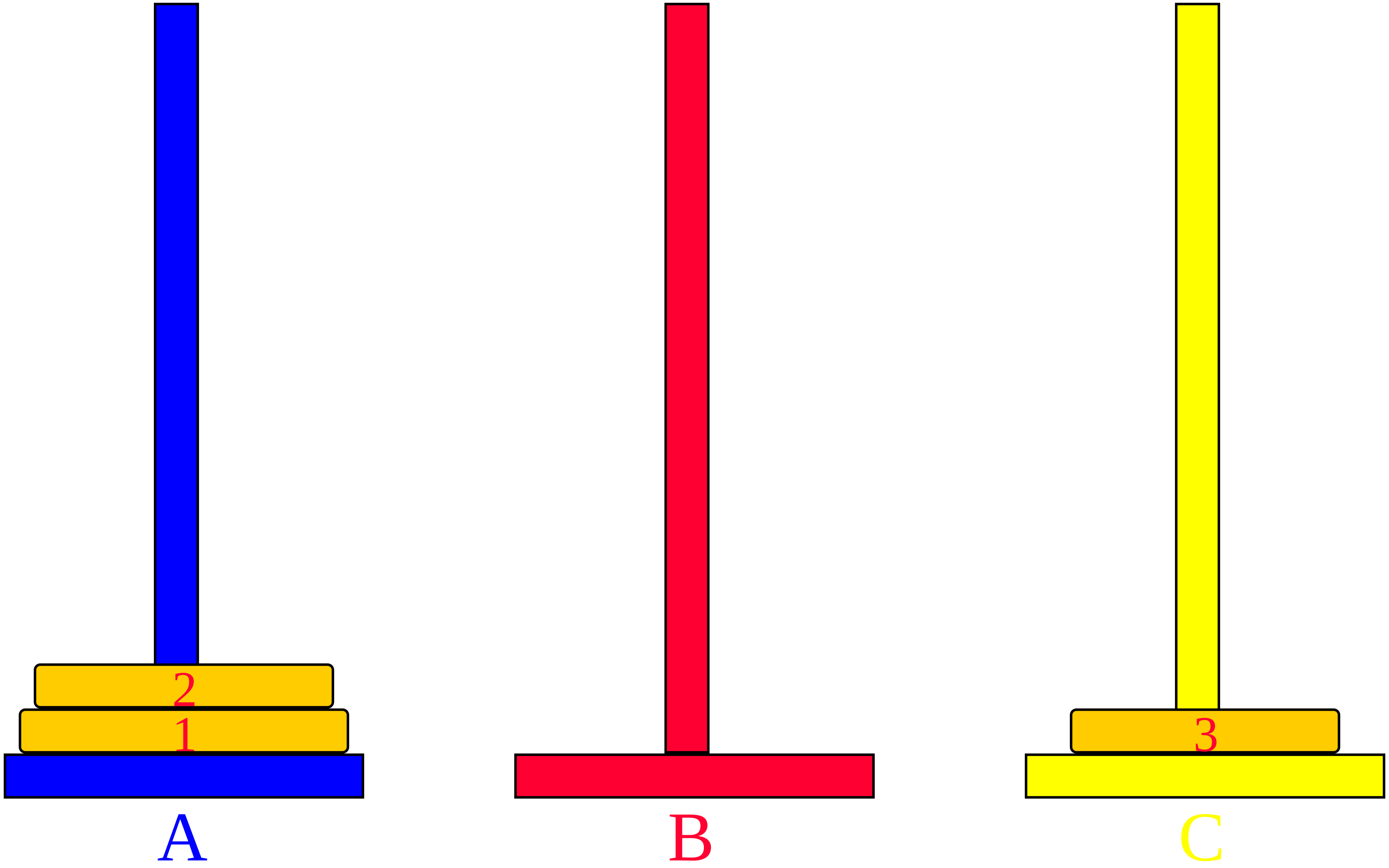
- 64 gold disks to be moved from tower A to tower C
- each tower operates as a stack
- cannot place big disk on top of a smaller one

# Towers Of Hanoi/Brahma



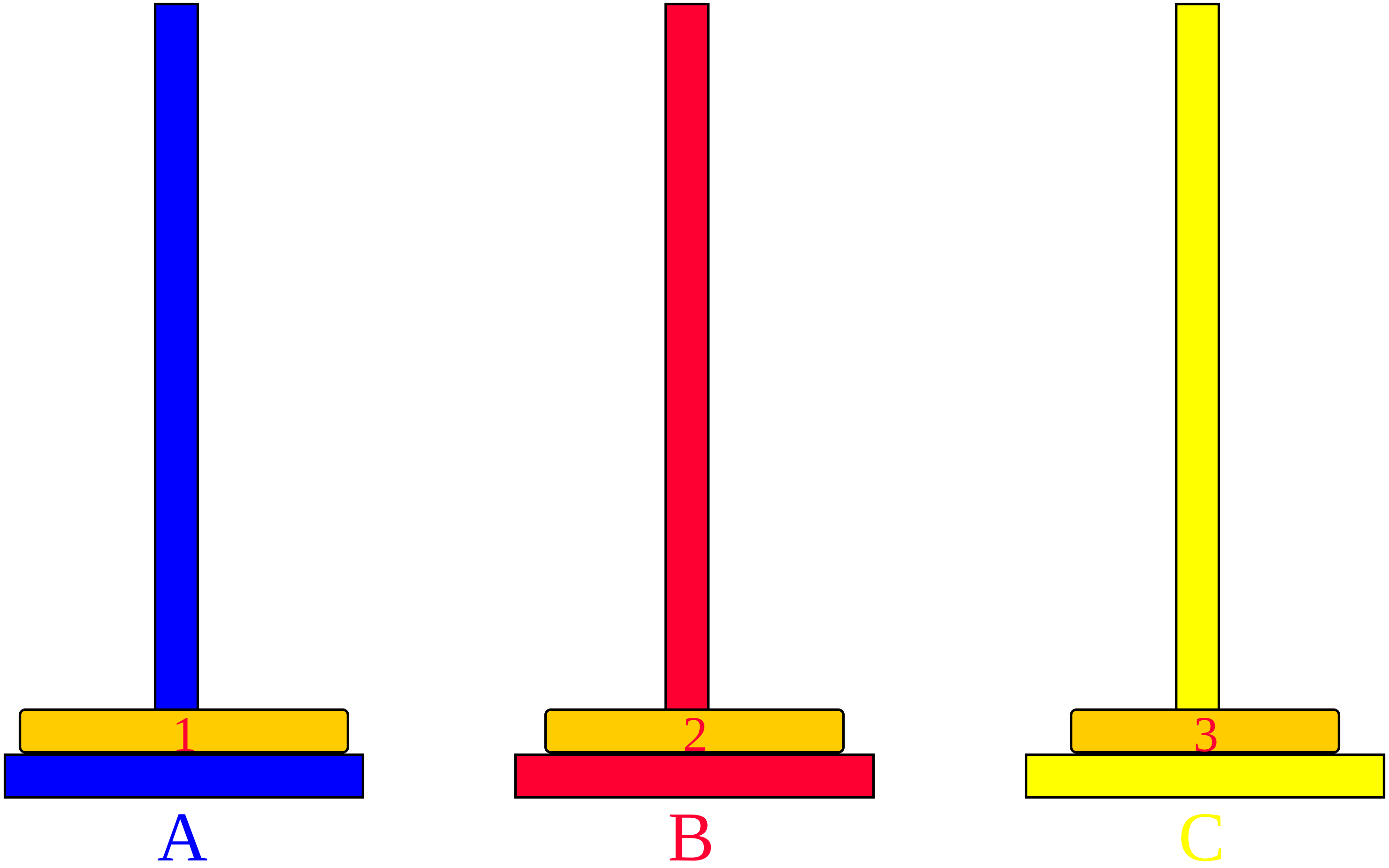
- 3-disk Towers Of Hanoi/Brahma

# Towers Of Hanoi/Brahma



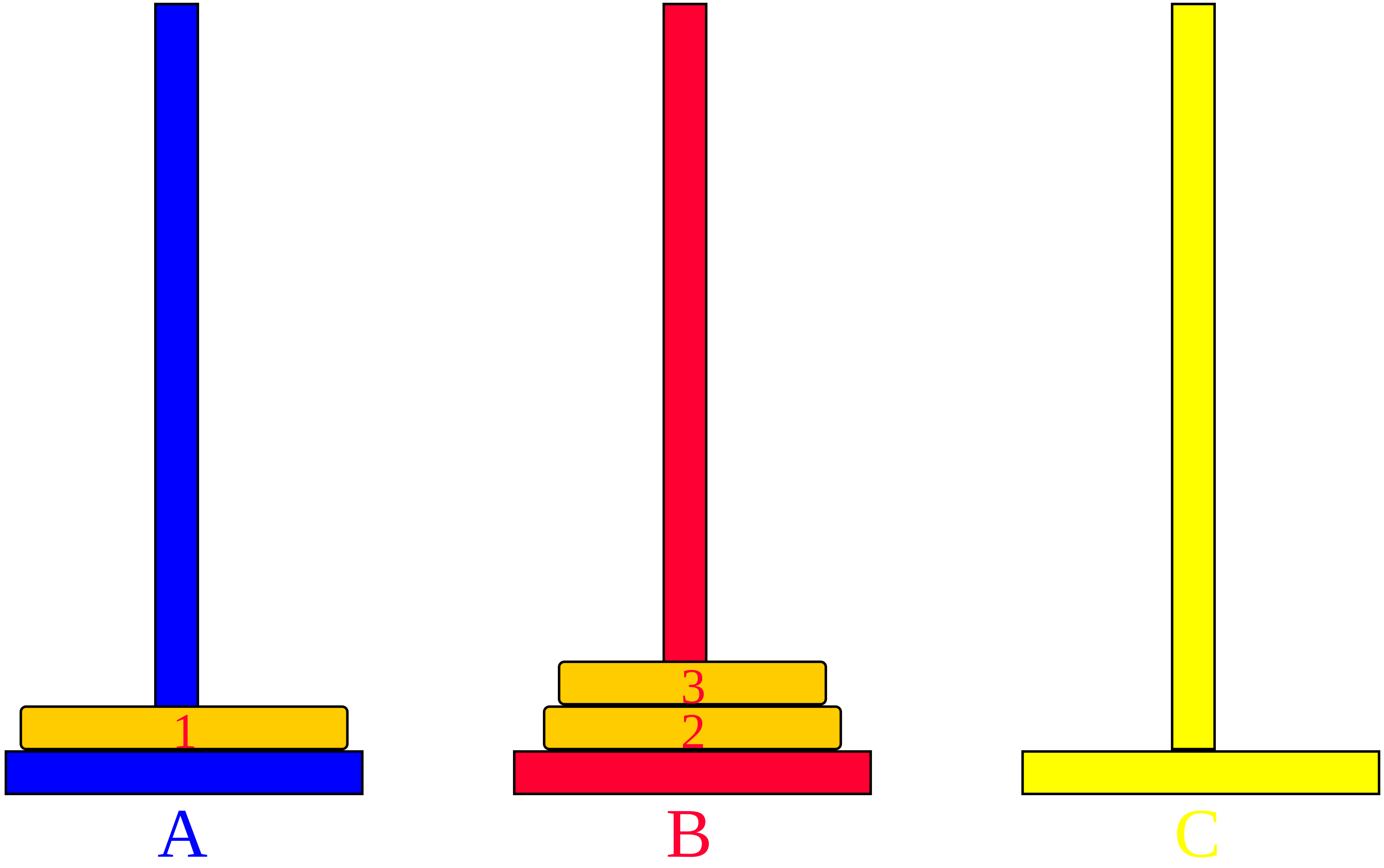
- 3-disk Towers Of Hanoi/Brahma

# Towers Of Hanoi/Brahma



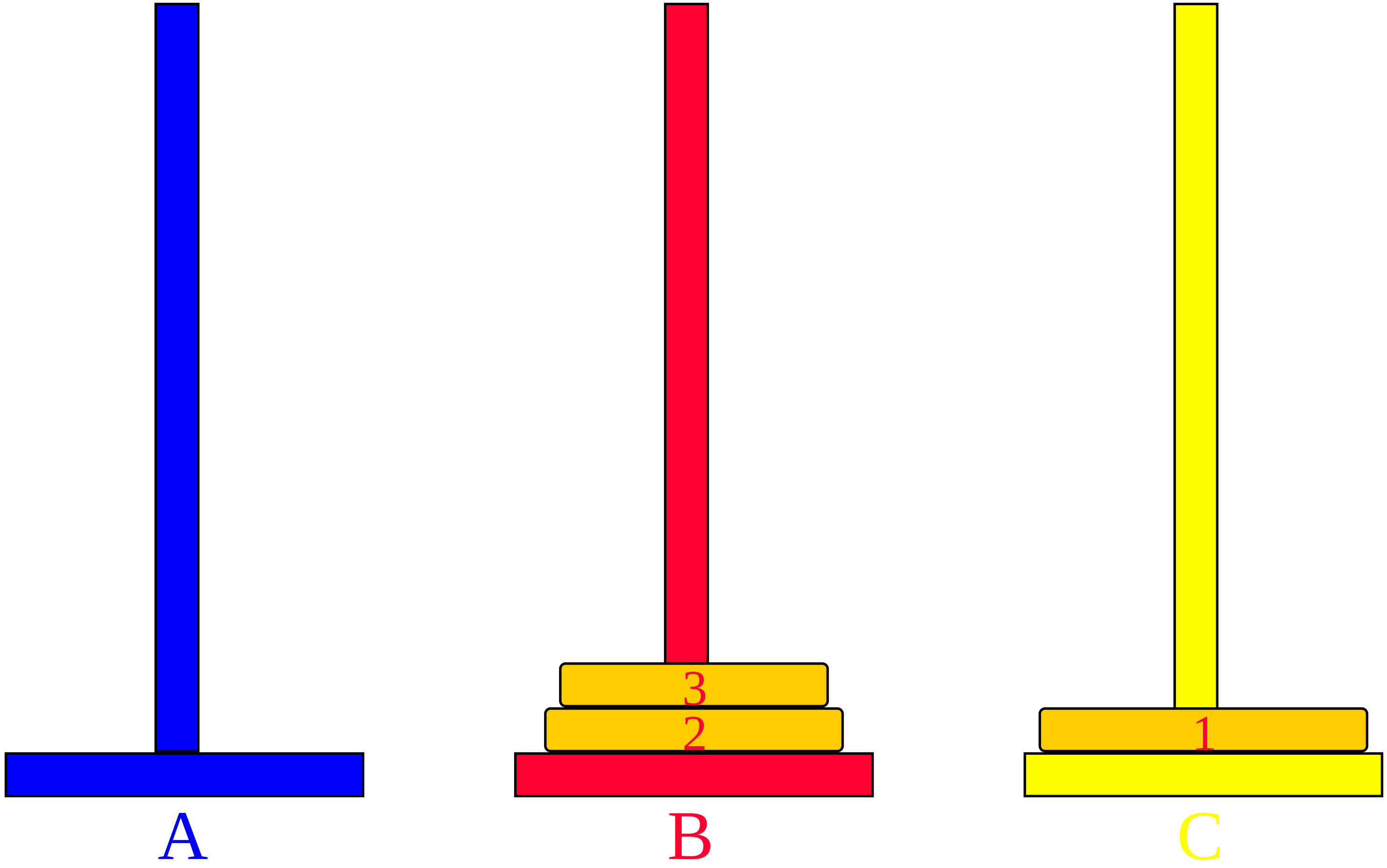
- 3-disk Towers Of Hanoi/Brahma

# Towers Of Hanoi/Brahma



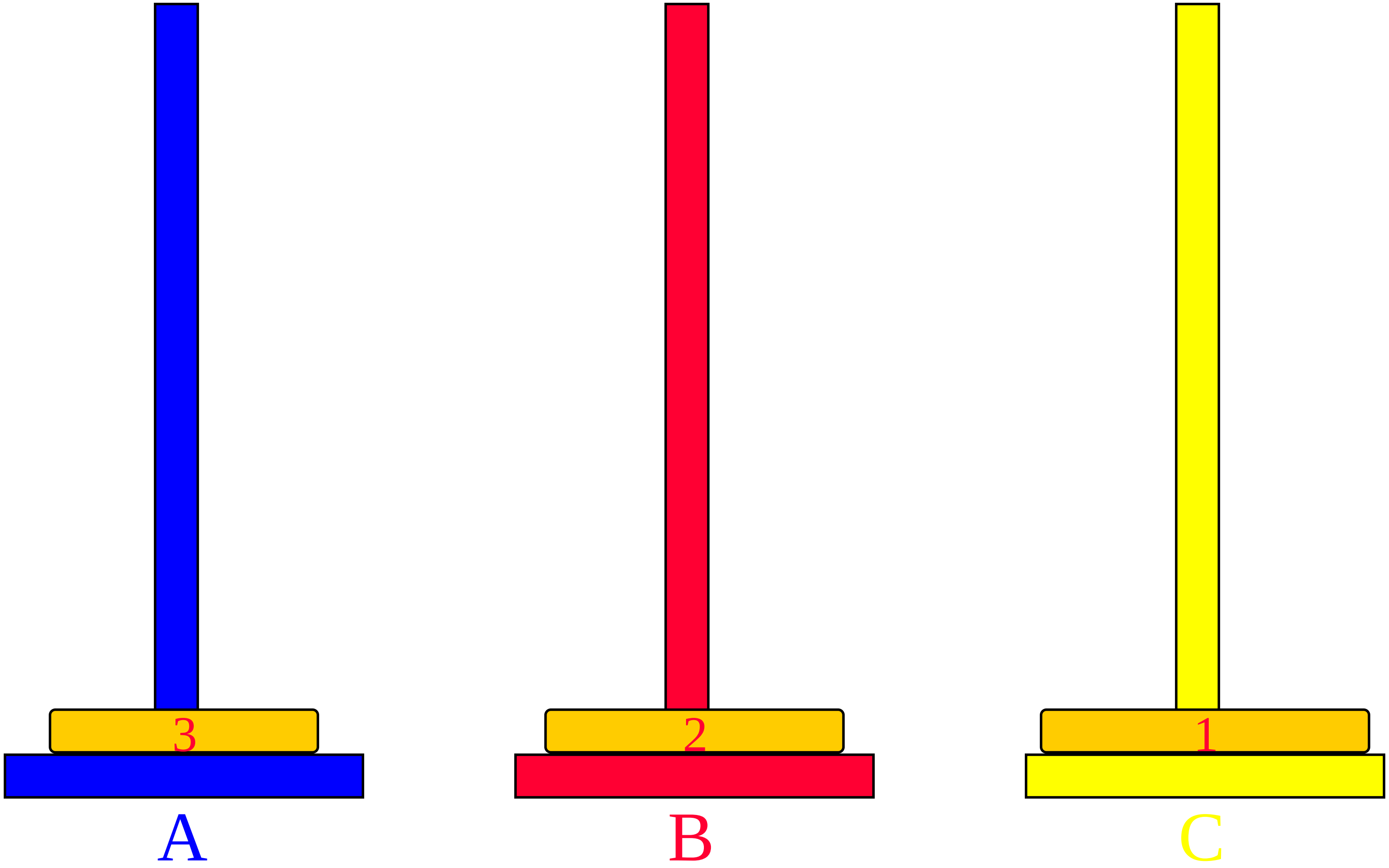
- 3-disk Towers Of Hanoi/Brahma

# Towers Of Hanoi/Brahma



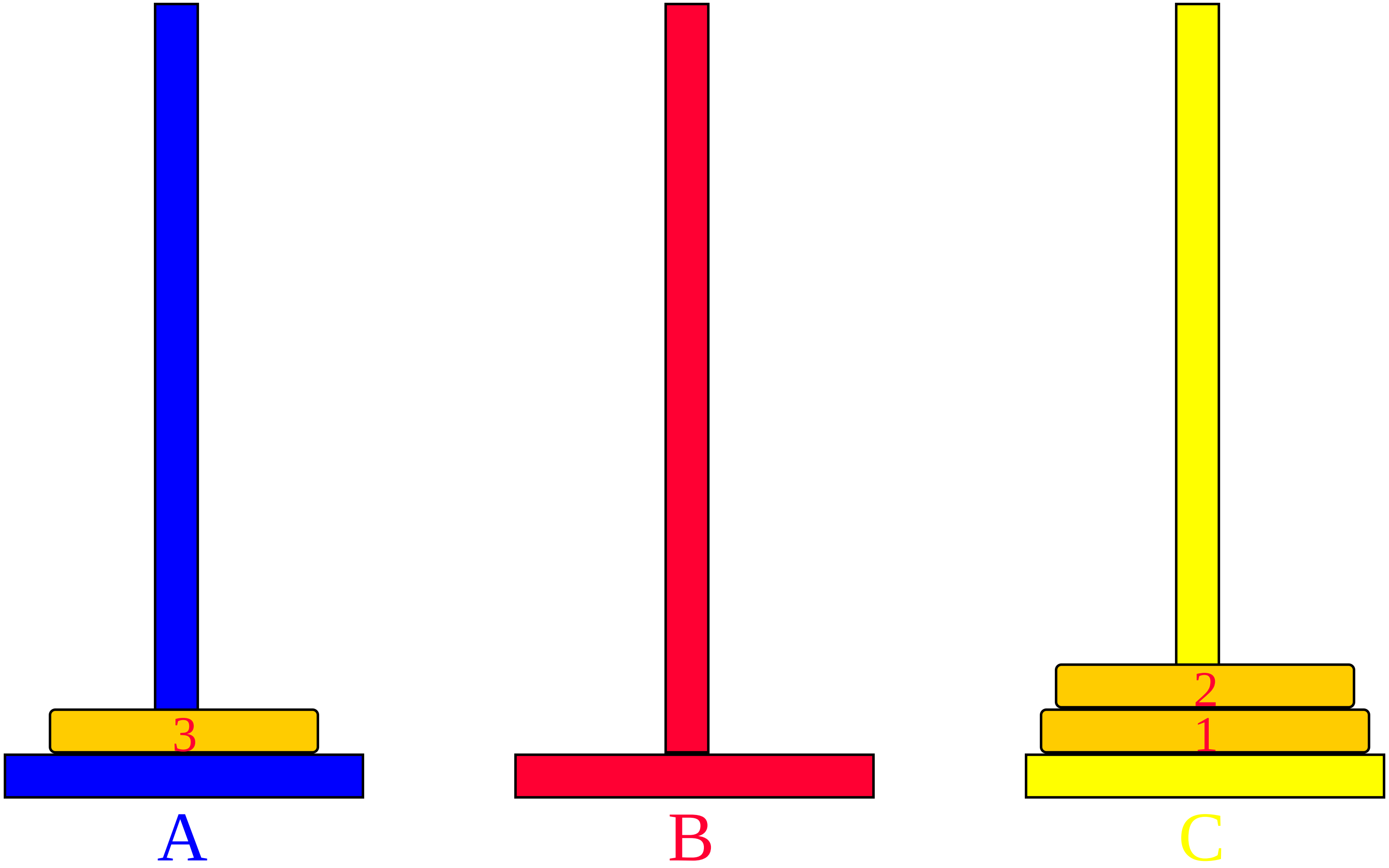
- 3-disk Towers Of Hanoi/Brahma

# Towers Of Hanoi/Brahma



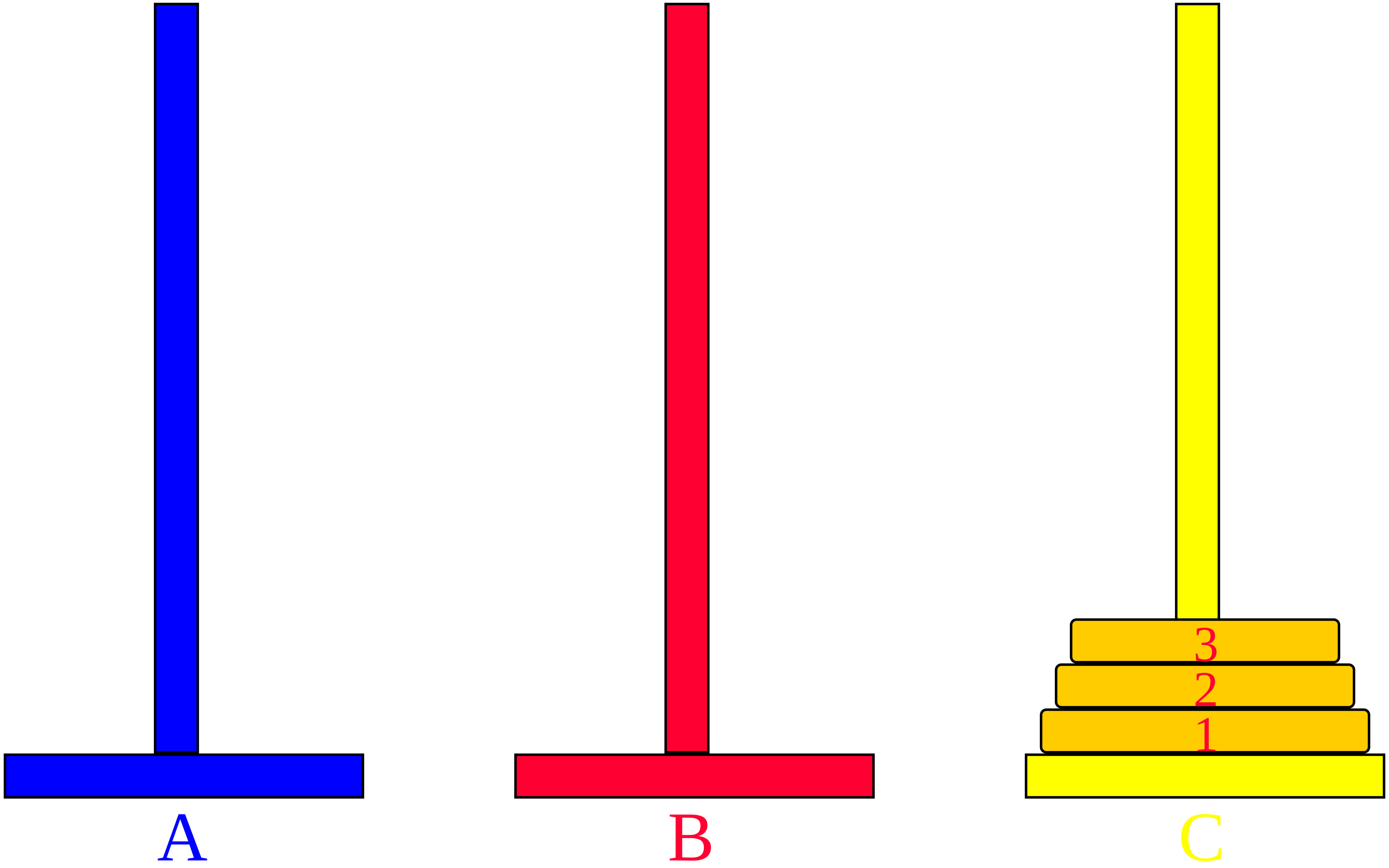
- 3-disk Towers Of Hanoi/Brahma

# Towers Of Hanoi/Brahma



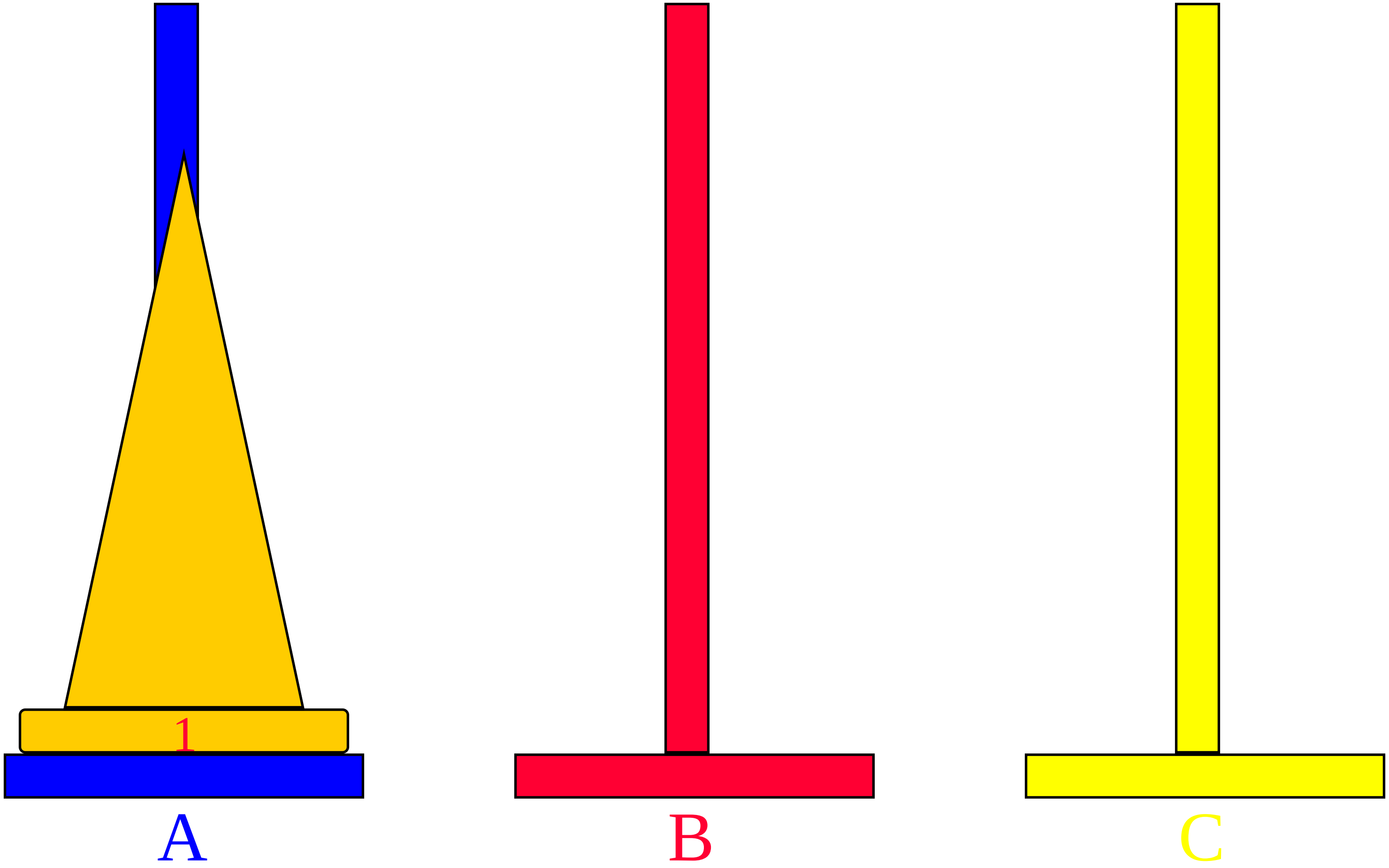
- 3-disk Towers Of Hanoi/Brahma

# Towers Of Hanoi/Brahma



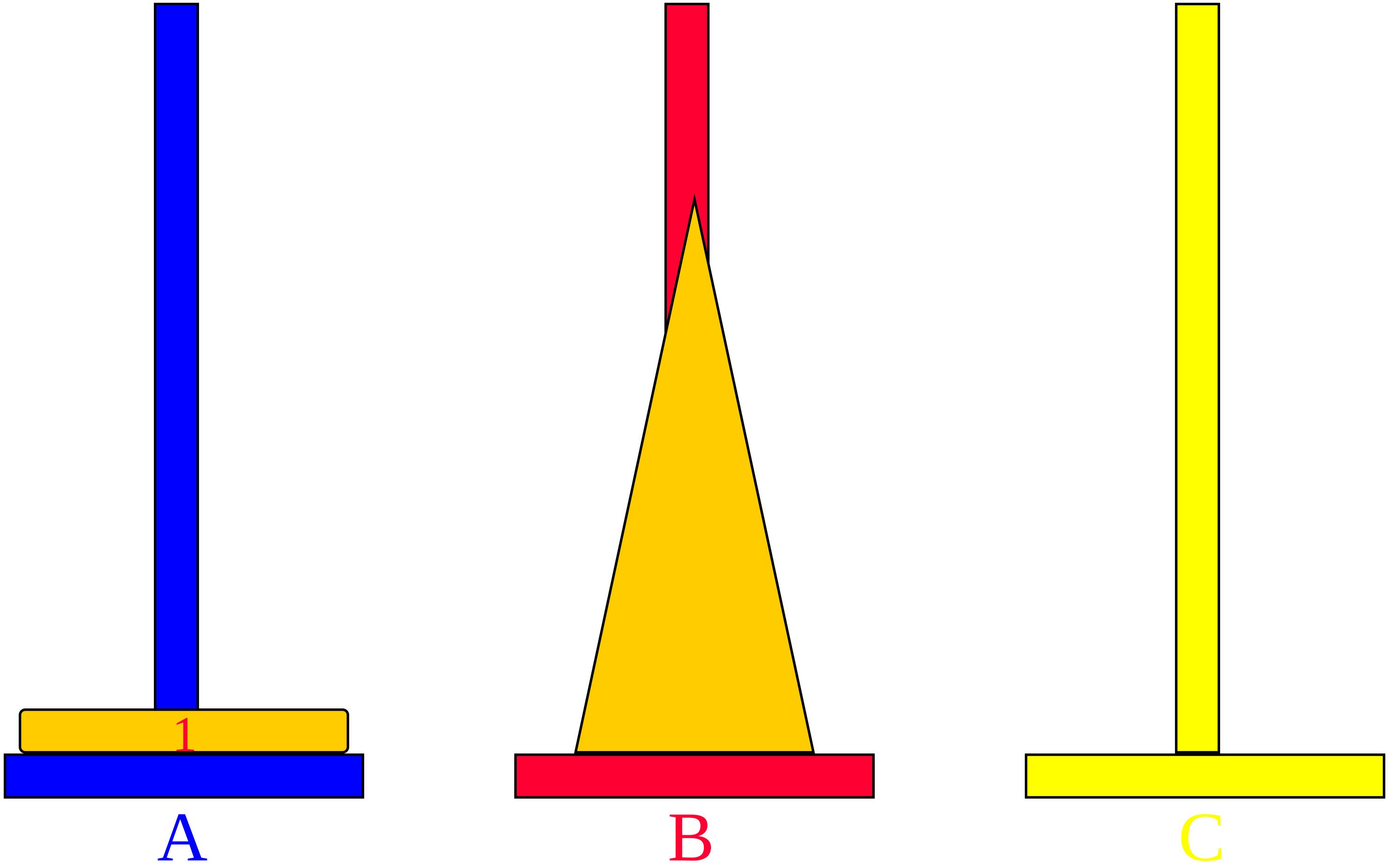
- 3-disk Towers Of Hanoi/Brahma
- 7 disk moves

# Recursive Solution



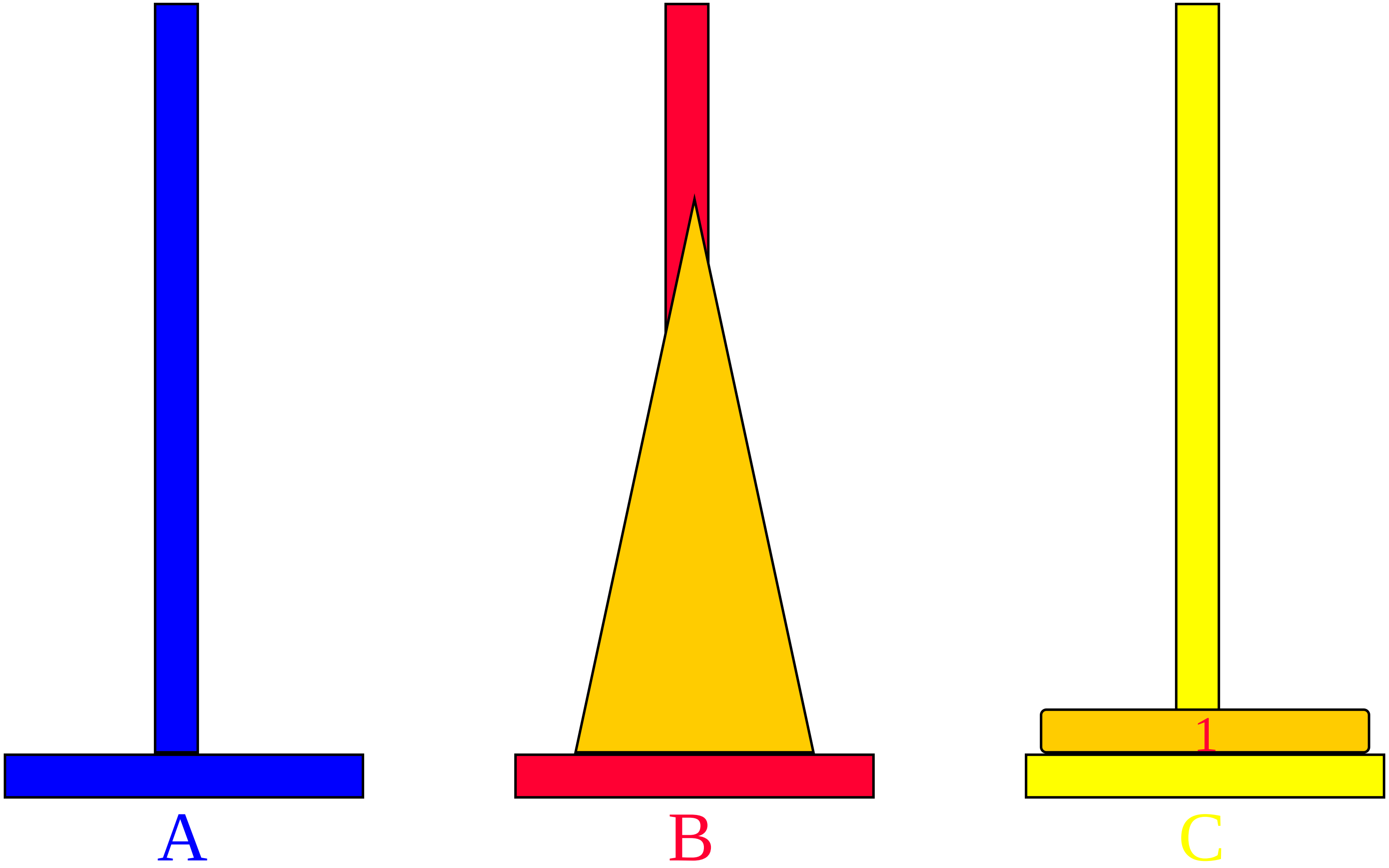
- n > 0 gold disks to be moved from A to C using B
- move top n-1 disks from A to B using C

# Recursive Solution



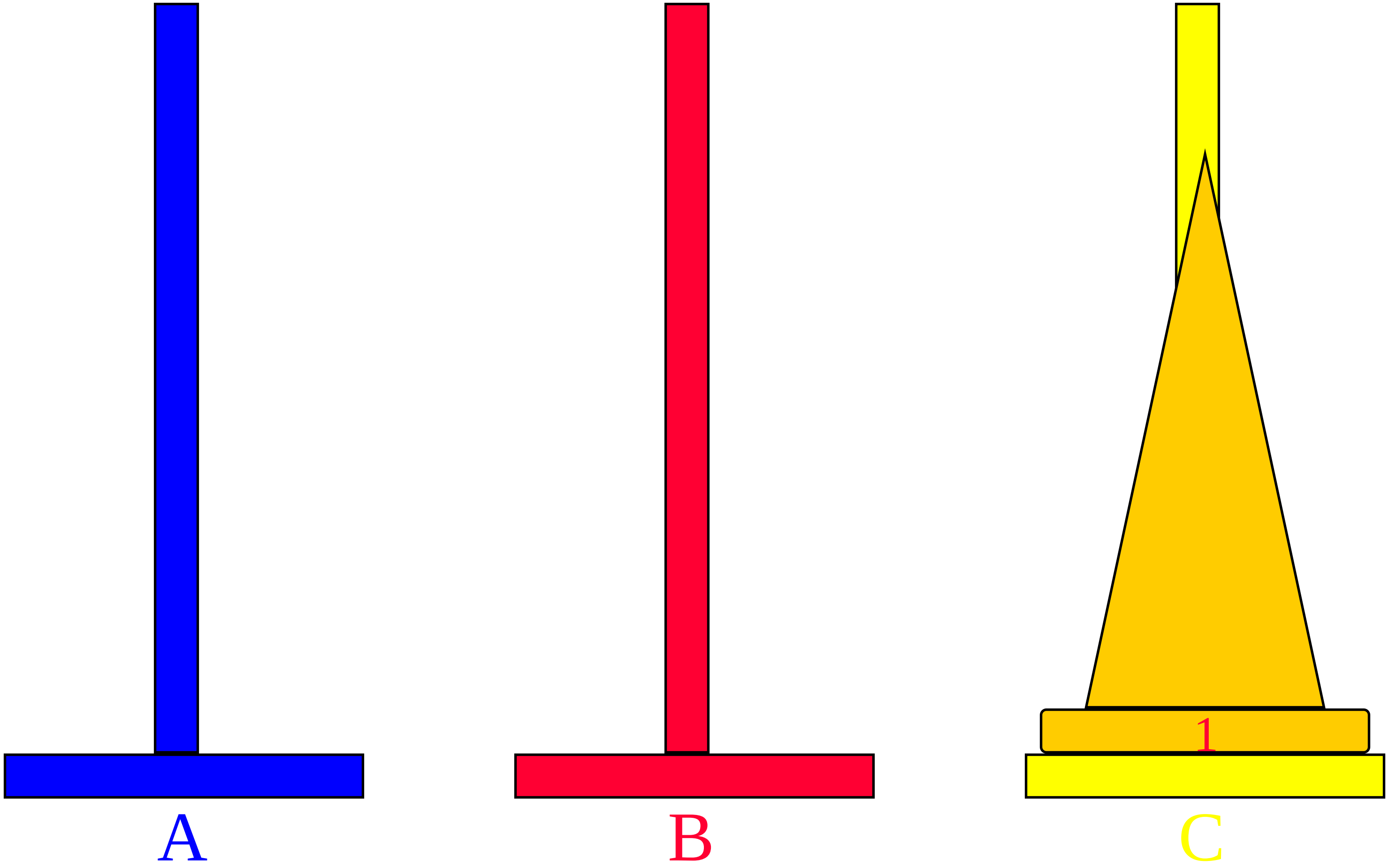
- move top disk from A to C

# Recursive Solution



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

# Recursive Solution



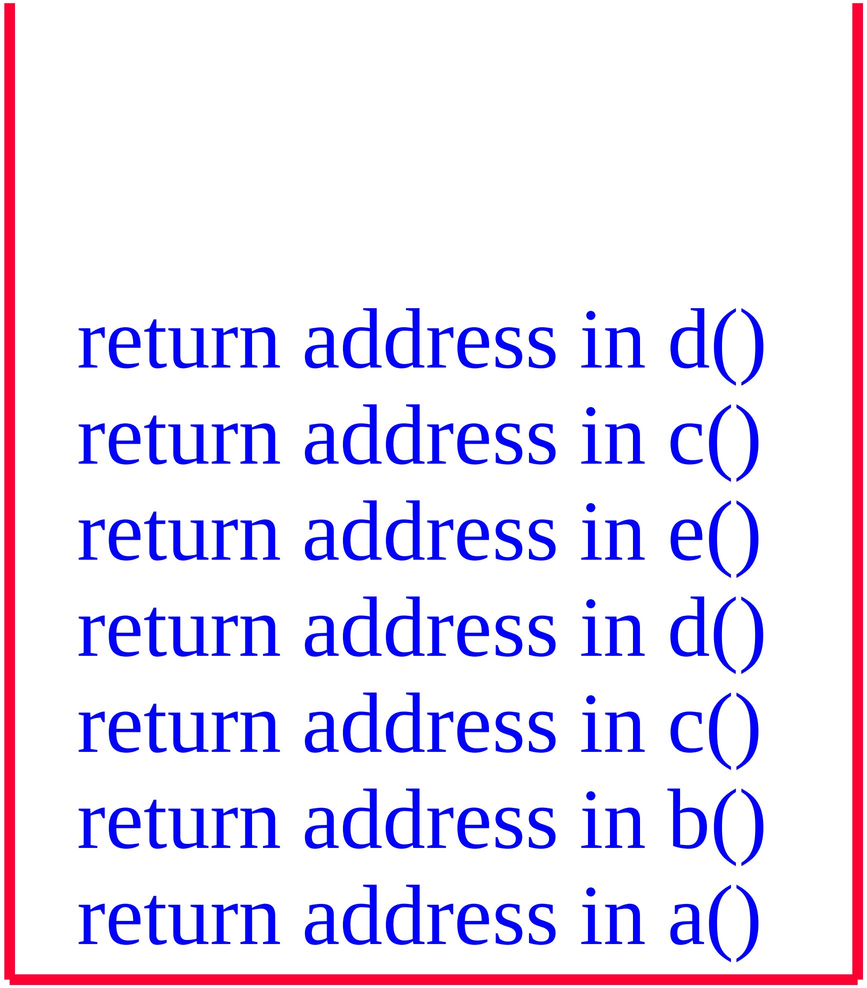
- moves(n) = 0 when n = 0
- moves(n) = 2*moves(n-1) + 1 = $2^n$-1 when n > 0

# Towers Of Hanoi/Brahma

- moves(64) = 1.8 * $10^{19}$ (approximately)
- Performing $10^9$ moves/second, a computer would take about 570 years to complete.

# Method Invocation And Return

```
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()

# Try-Throw-Catch

- When you enter a try block, push the address of this block on a stack.
- When an exception is thrown, pop the try block that is at the top of the stack (if the stack is empty, terminate).
- If the popped try block has no matching catch block, go back to the preceding step.
- If the popped try block has a matching catch block, execute the matching catch block.

# Decimal to binary

```
DecimaltoBinary()
{  createStack(S)                   // Stack S=new Stack()
   while(number != 0)
         remainder = number % 2
         S.push(remainder)
         number = number / 2
   while(not Empty(S))
        x= S.pop()
         print(x)
    }
```

# Other examples

- Traversing in tree
- Solving expressions
- Find shortest path (in graphs)
- ….

# Questions