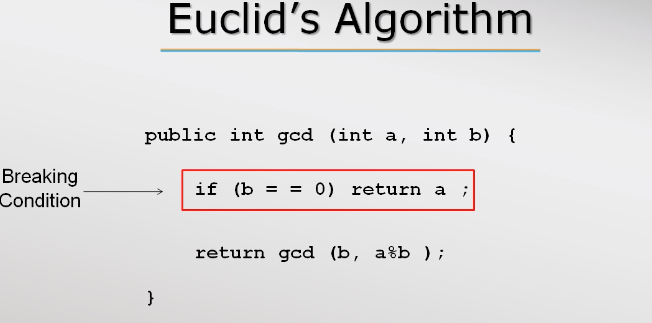
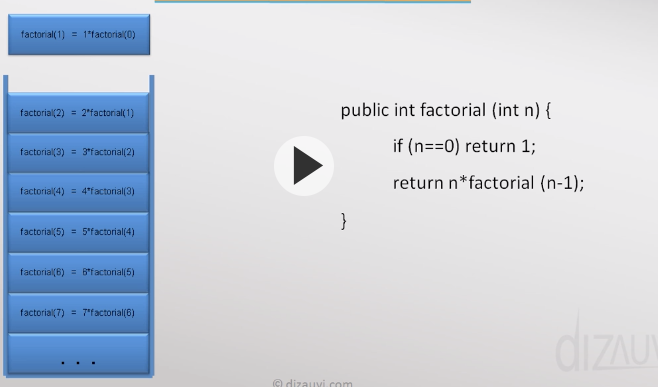
Recursive algorithms call themselves; they must have a breaking condition. If no breaking condition then there will be a stack overflow. Since stack frames are pushed on to the invocation stack but they are never popped off the invocation stack

**Examples:**

**Euclidean algorithm**, procedure for finding the [greatest common divisor](https://www.britannica.com/science/greatest-common-divisor) (GCD) of two numbers



**Factorial algorithm**



Program creates a method stack (AKA call stack). This stack (data structure) keeps track of the active methods. Stack contains stack frames that contain methods, local variables, and reference variables

factorial(7) is called and this returns 7\*factorial(6), since factorial(6) cannot return

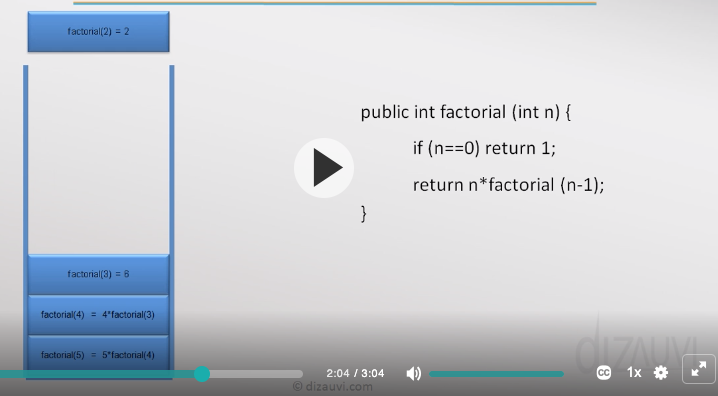
factorial(7) = 7\*factorial(6) is pushed on to the stack

next factorial(6) is called and this returns 6\*factorial(5), since factorial(5) cannot return

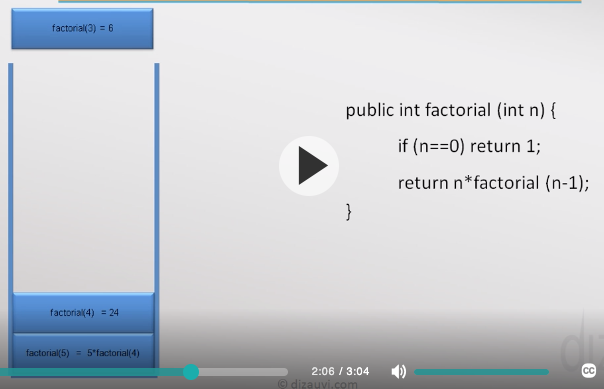
factorial(5) = 6\*factorial(5) is pushed on to the stack

…. and so on.

Stack frames can be pooped and return values evaluated…



Stack frames can be pooped and return values evaluated…



Until all stack frames have been popped. Value of expression of last stack frame is result of the recursive method call.

**Stackoverflow**

Stack overflow can occur under 2 conditions:

* No breaking condition
* The stack is too deep, i.e. if n too is large in example above

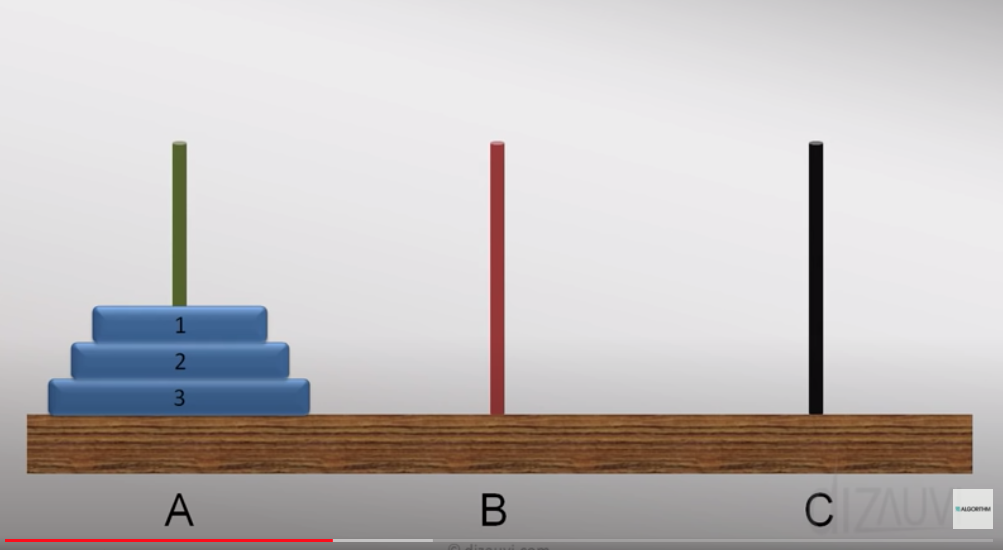
**Tail recursion** can be used to avoid stackoverflow errors

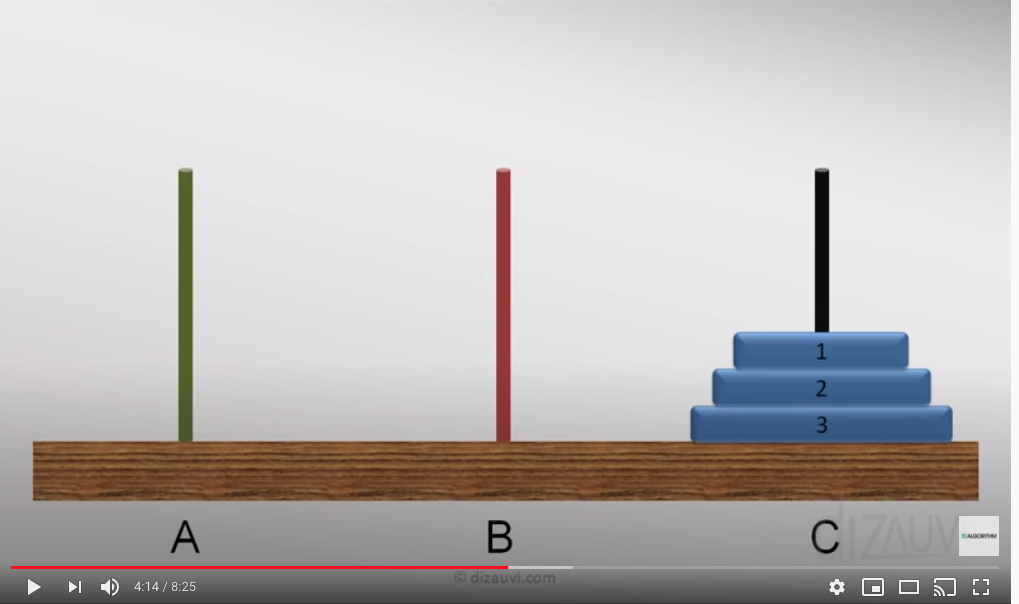
**Hanoi Tower**

The idea is to move all discs to A to C using B as an intermediary, the order must be maintained in doing so, bigger disc must never be on top of a smaller one. Only move one disc at a time. There can be many disks

<https://www.youtube.com/watch?v=fffbT41IuB4>

With 3 disks





**Solution:**

Moving disc 1 from A to C

Moving disc 2 from A to B

Moving disc 1 from C to B

Moving disc 3 from A to C

Moving disc 1 from B to A

Moving disc 2 from B to C

Moving disc 1 from A to C

**General Solution**

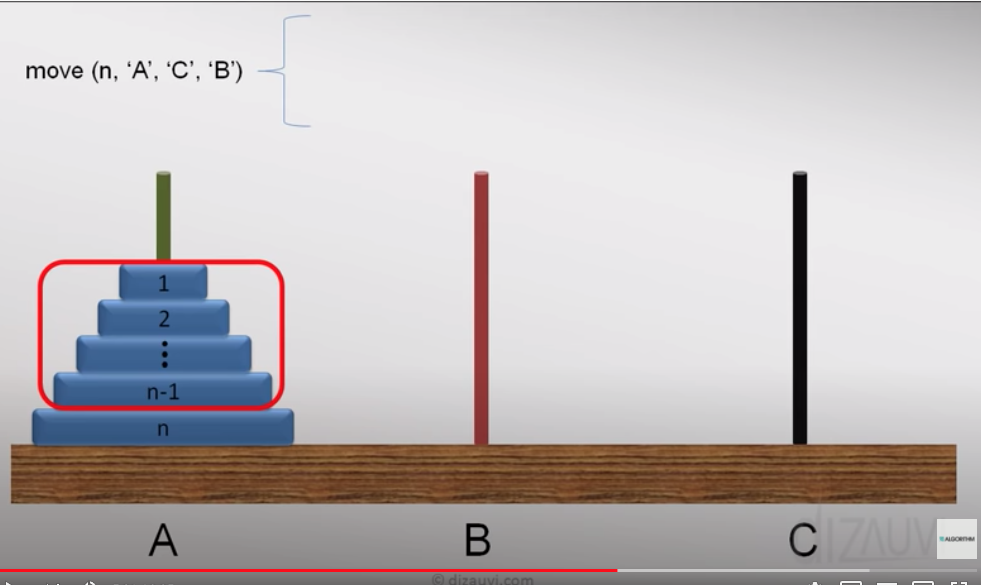
n – number of disks

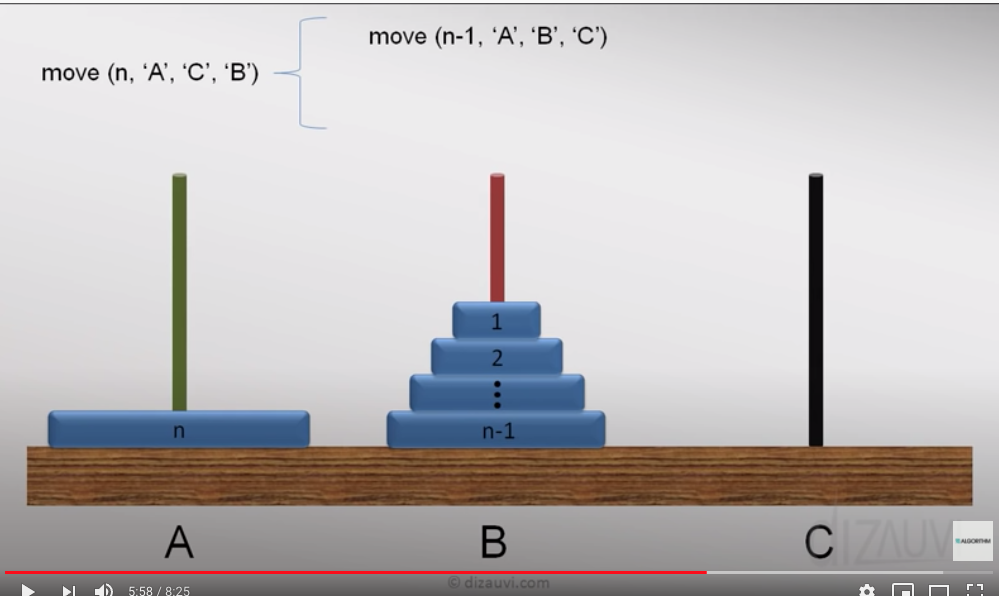
‘A’ – from tower

‘C’ – to tower

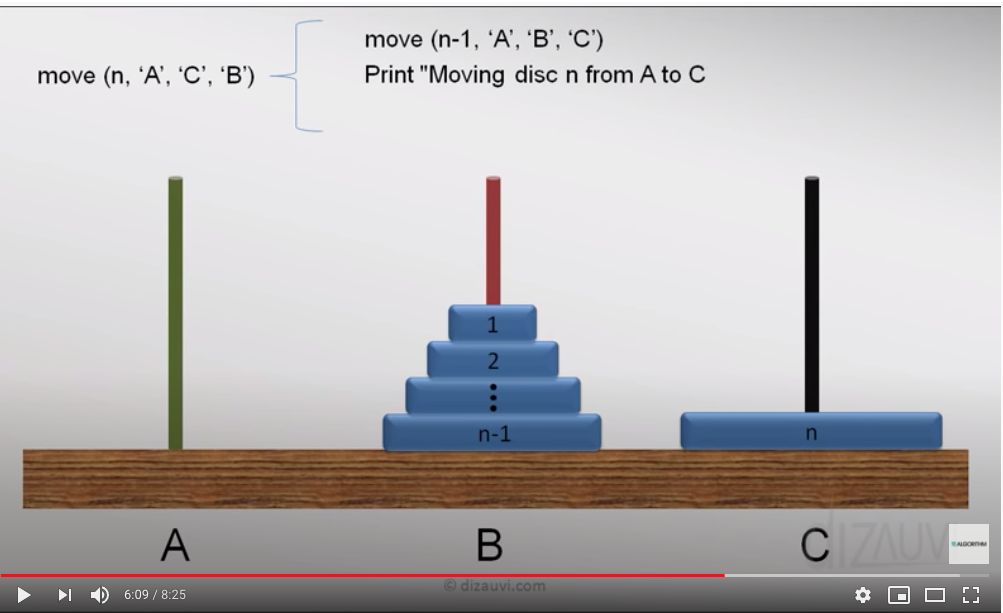
‘B’ – intermediate tower

First convince ourselves that we can move all disks from one tower to another, using via the intermediary tower, i.e from tower ‘A’ to tower ‘C’ via ‘B’

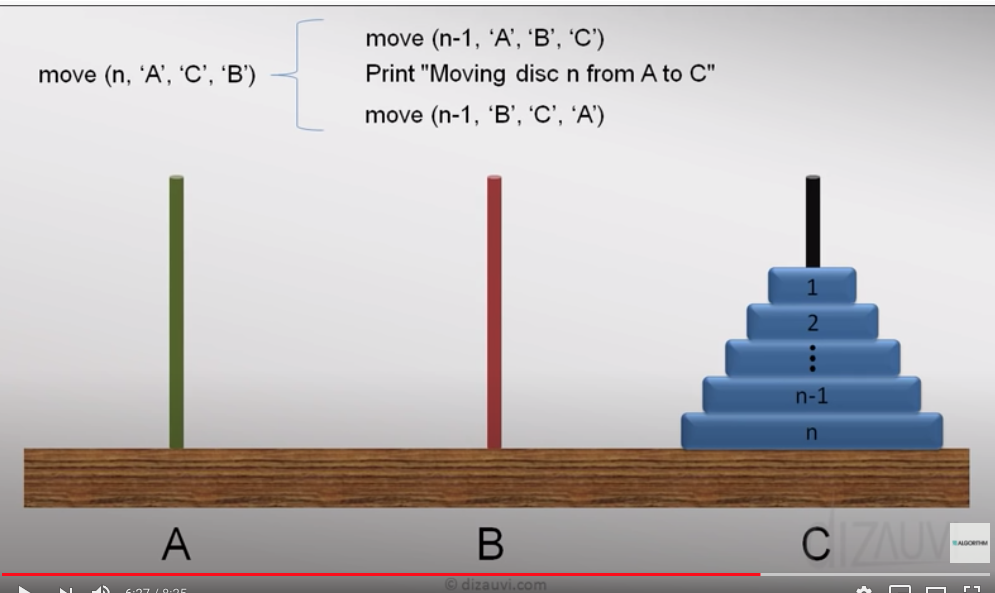


If we convinced our self of above then we should be able to move n-1 discs from tower ‘A’ to tower ‘B’ via ‘C’

Then we can move disc n from ‘A’ t0 ‘C’ and print this out



Then move n-1 discs from tower ‘B’ to tower ‘C’ using ‘A’



public class TowerOfHanoi {  
  
 public void move(int numberOfDiscs, char from, char to, char inter) {  
 if (numberOfDiscs == 1) {  
 System.*out*.println("Moving disk 1 from " + from + " to " + to);  
 } else {  
 move(numberOfDiscs - 1, from, inter, to);  
 System.*out*.println("Moving disc " + numberOfDiscs + " from " + from + " to " + to); // this is moving disc n from A to C  
 move(numberOfDiscs - 1, inter, to, from);  
 }  
 }  
  
 public static void main(String[] args) {  
 TowerOfHanoi towerOfHanoi = new TowerOfHanoi();  
 towerOfHanoi.move(3, 'A', 'C', 'B');  
 }  
}