

lecture 10

recursion

- $n!$ (sketch)
- Fibonacci
- Towers of Hanoi

Recall : factorial

definition $n! = 1 \cdot 2 \cdot 3 \dots (n-1) \cdot n$

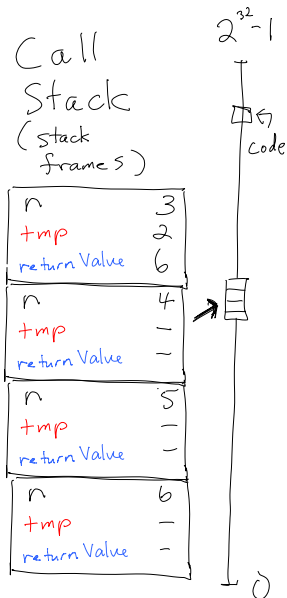
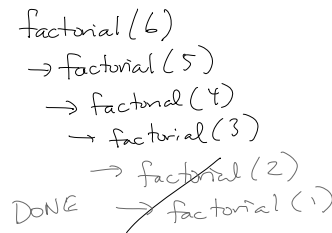
```
factorial (n) { // n ≥ 1
    if n == 1
        return 1
    else
        return n * factorial (n-1)
}
```

Another way to think of it.

```
int factorial (n) {
    int returnValue
    int tmp
    if n == 1
        returnValue ← 1
    else {
        tmp ← factorial (n-1)
        returnValue ← n * tmp
    }
}
```

returnValue is a special "local variable" which holds the int value that is returned.

```
int factorial (n) {
    int returnValue
    int tmp
    if n == 1
        returnValue ← 1
    else {
        tmp ← factorial (n-1)
        returnValue ← n * tmp
    }
}
```



- you will learn in detail how the call stack works in COMP 273 (or ESCE 221)

- you can inspect current stack frame using the Eclipse debugger (demo)

Example : Fibonacci numbers

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n+2) = F(n+1) + F(n), \quad n \geq 0$$

n	0	1	2	3	4	5	6	7	8	9	10	11	...
F(n)	0	1	1	2	3	5	8	13	21	34	55	89	...

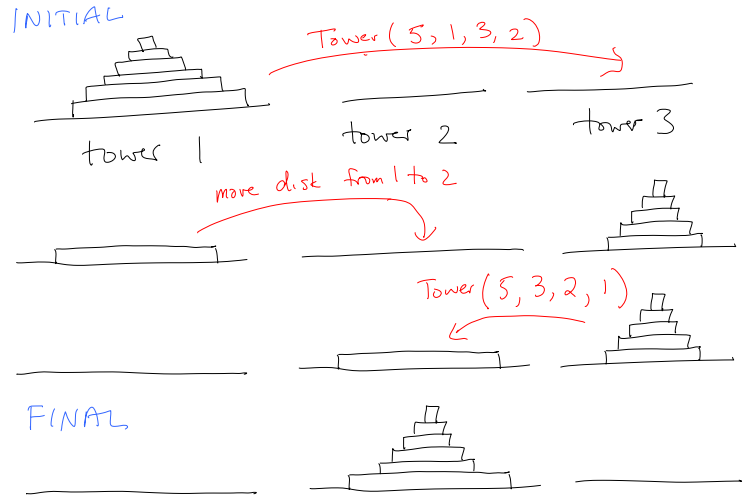
This suggests a non-recursive algorithm.

Recursive Algorithm

```

Tower (n, start, finish, other) {
  if n > 0 {
    Tower (n-1, start, other, finish)
    Move disk from start to finish
    Tower (n-1, other, finish, start)
  }
}
    
```

Example: Tower(6, 1, 2, 3)



```

Tower(6, 1, 2, 3)
  → Tower(5, 1, 3, 2)
    → Tower(4, 1, 2, 3)
      Move disk from 1 to 3
      Tower(4, 3, 2, 1)
        Move disk from 1 to 2
        Tower(5, 3, 2, 1)
          → Tower(4, 3, 1, 2)
            move disk from 3 to 2
            Tower(4, 1, 2, 3)
    
```

Some Questions you might have

- is it correct?
- how is it implemented? (call stack)
- how many "moves" does Tower(n, *, *.*) require?

"Tower(n, start, finish, other) is correct for any n."

Correct means doesn't break rule that smaller disks are on larger disks.

Proof by induction

Base case: n=1 ✓

Induction step: (CONVINCE YOURSELF)

if Tower(k, *, *, *) is correct
then Tower(k+1, *, *, *) is correct ✓

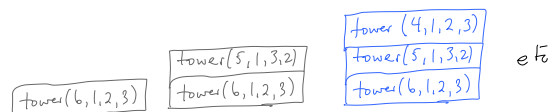
Call Stack

```

tower(n, a, b, c) {
  if n > 0 {
    tower(n-1, a, c, b)
    move disk from a to b
    tower(n-1, c, b, a)
  }
}
    
```

Code (instructions) →

Each stack frame contains
- n
- a, b, c
- ...

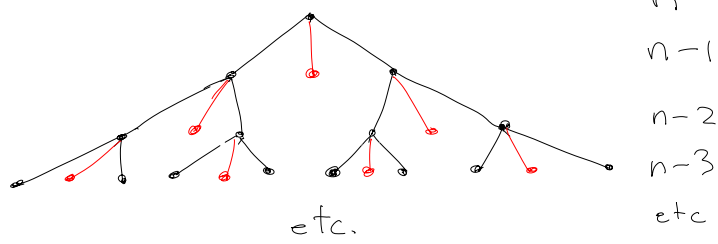


call stack evolves over time

32
2-1

0

How many disk moves . ?



By inspection :

$$1 + 2 + 4 + 8 + 2^{n-1} = 2^n - 1$$