DATA STRUCTURES & ALGORITHMS

Recursion

Instructor: Engr. Laraib Siddiqui

Recurrence

Basic objective

- When we have a bigger problem whose solution is complex.
- We decompose the problem into smaller units until we reach to the smallest sub-problem (base case) for which a solution is known.
- Then go back in reverse order and build upon the solutions of the sub-problems.

Recursion

A well defined mathematical function in which the function being defined is applied within its own definition.

Recursive Functions

- Function may call itself
- Function may call other Function and
- the other function in turn again may call the calling Function

Direct & indirect Recursion

Recursion is said to be *direct* when functions calls *itself* directly and is said to be *indirect* when it calls *other function* that in turn calls it

Other Types

Linear recursion : makes at most one recursive call each time it is invoked.

Binary recursion : makes two recursive calls.

Multiple recursion : method may make (potentially more than two) recursive calls.

Finding a recursive solution

- Each successive recursive call should bring you closer to a situation in which the answer is known
- A case for which the answer is known (and can be expressed without recursion) is called a base case
- Each recursive algorithm must have at least one base case, as well as the general recursive case

Recursion vs. Iteration

The factorial of a positive integer

For example, $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$.

Iterative Solution

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 & \text{if } n \geq 1 \end{cases}$$

Recursive Solution

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \text{factorial}(n-1) & \text{if } n \geq 1 \end{cases}$$

Example

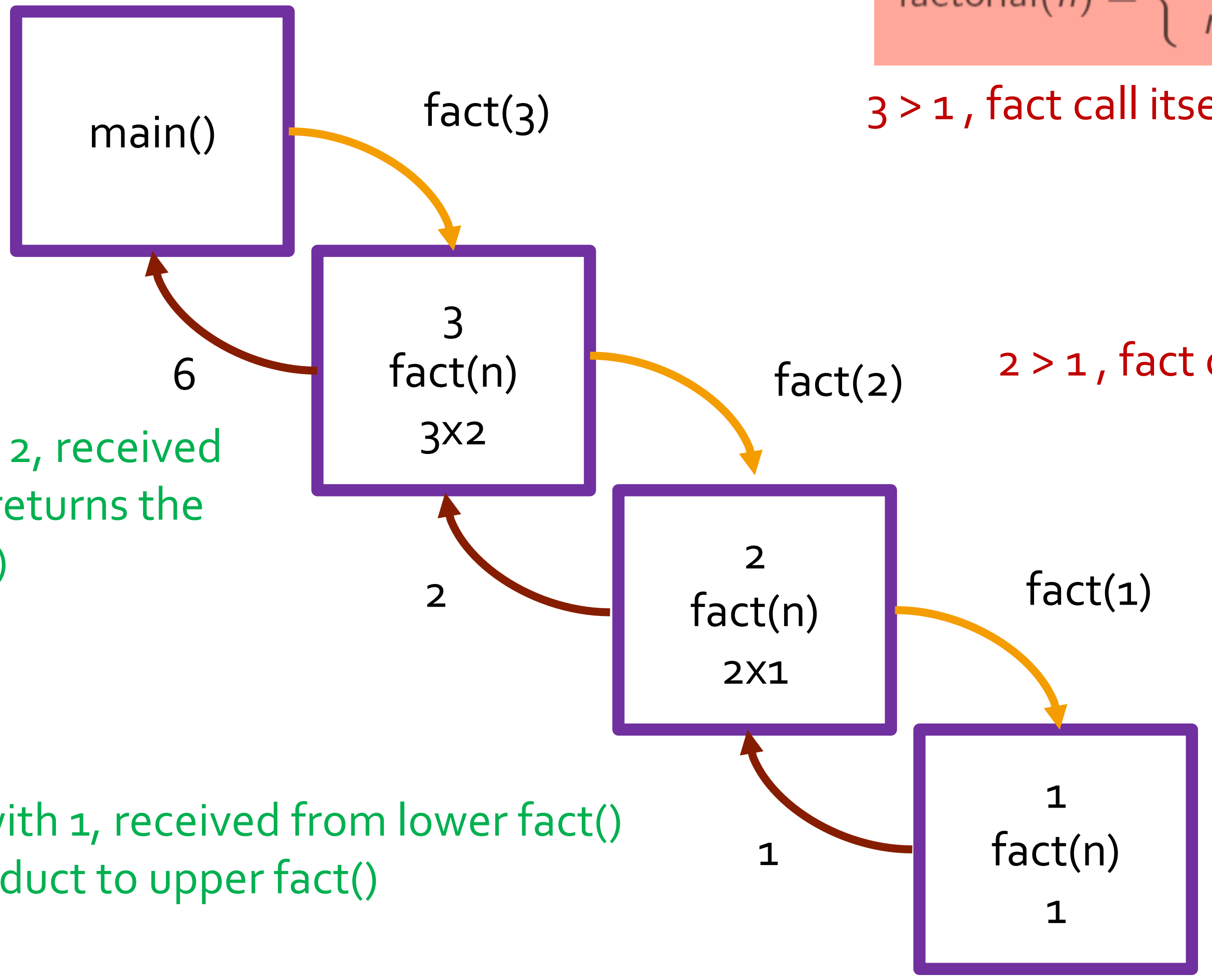
Factorial of n

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * \text{factorial}(n-1) & \text{if } n \geq 1 \end{cases}$$

Example - Factorial

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * \text{factorial}(n-1) & \text{if } n \geq 1 \end{cases}$$

3 > 1 , fact call itself



2 > 1 , fact call itself

n=1, returns 1

fact() multiplies 3 with 2, received from lower fact() and returns the product to upper fact()

fact() multiplies 2 with 1, received from lower fact() and returns the product to upper fact()

Example Fibonacci Series

```
fib(n)
    if(n<2)
            return n
    else
            return ( fib(n-1) + fib(n-2) )
```

5

fib(5)

3 2

fib(4) + fib(3)

2 1 1 1

fib(3) + fib(2) fib(2) + fib(1)

1 1 1 0 1 0

fib(2) + fib(1) fib(1) + fib(0) fib(1) + fib(0)

1 0

fib(1) + fib(0)

n is 1 and 1 it will return n ,
after adding 1+1 it will return 2
to the upper function call and
so on..

n is 1 and 0 it will return n ,
after adding 1+0 it will return 1
to the upper function call

Example Tower of Hanoi

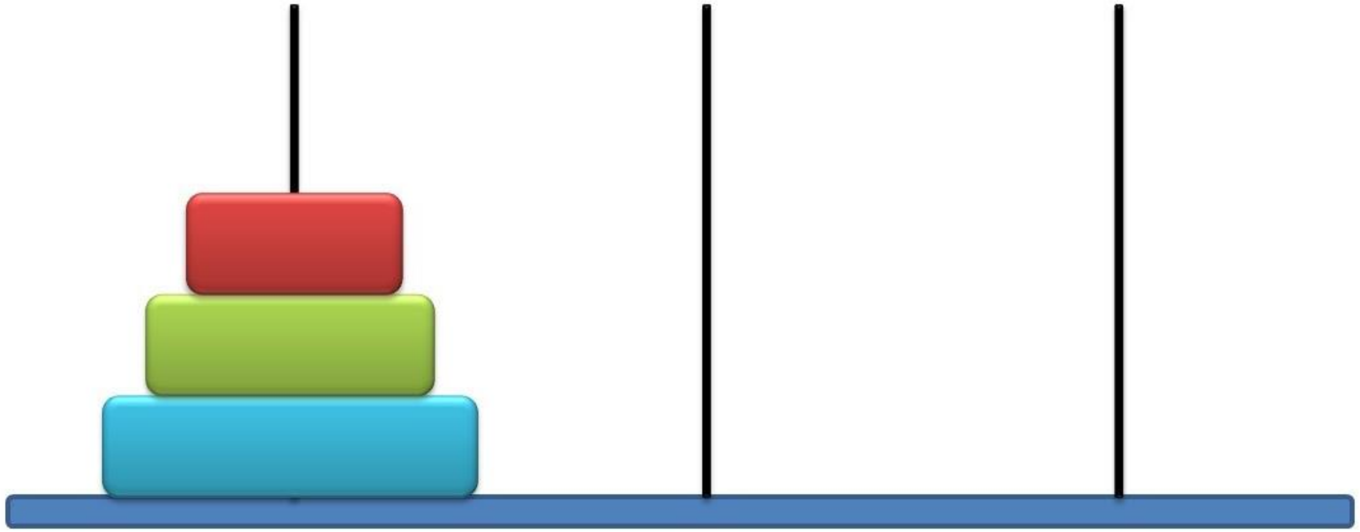
Objective

To transfer entire tower to one of the other pegs.

Rules

Only one disk may e moved at a time.

Lager disks can not be placed on the smaller disk.



Example Tower of Hanoi

Step 1: Move top(N-1) disks from Beg to Aux peg.

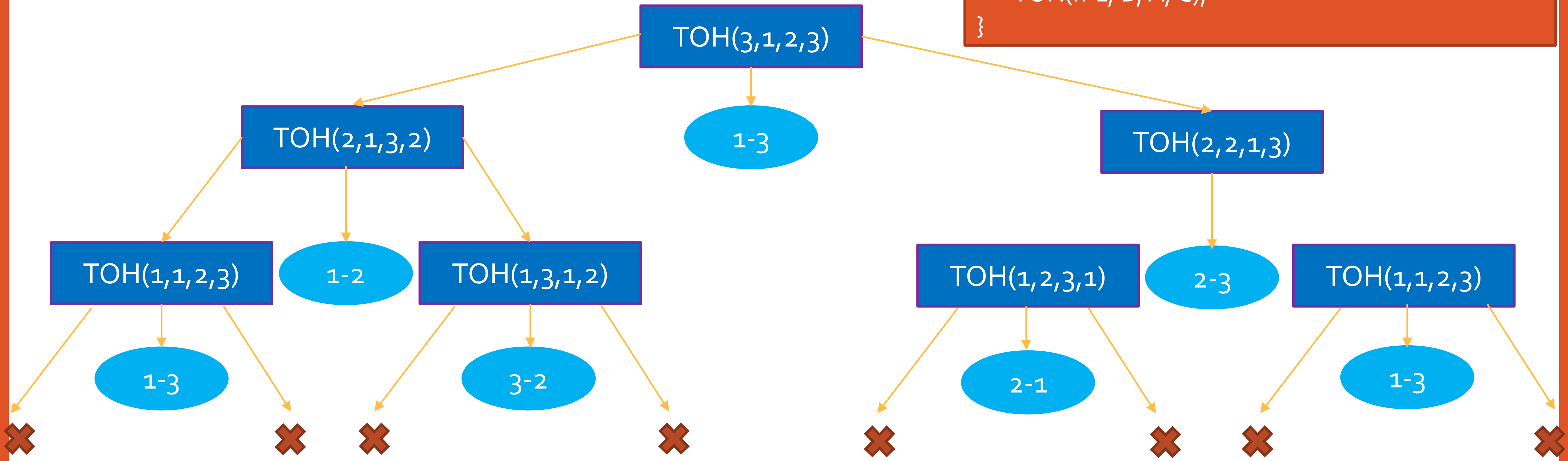
Step 2: Move 1 disk from Beg to End peg.

Step 3: Move top(N-1) disks from Aux to End peg.

- *T(N-1, Beg, End, Aux)*
- *T(1, Beg, Aux, End)*
- *T(N-1, Aux, Beg, End)*

Example Tower of Hanoi

```
void TOH(int n,int A, int B, int C)
{
    if(n>0)
     TOH(n-1, A,C,B);
     printf(“Move a Disc from %d to %d”, A,C);
     TOH(n-1, B, A, C);
}
```

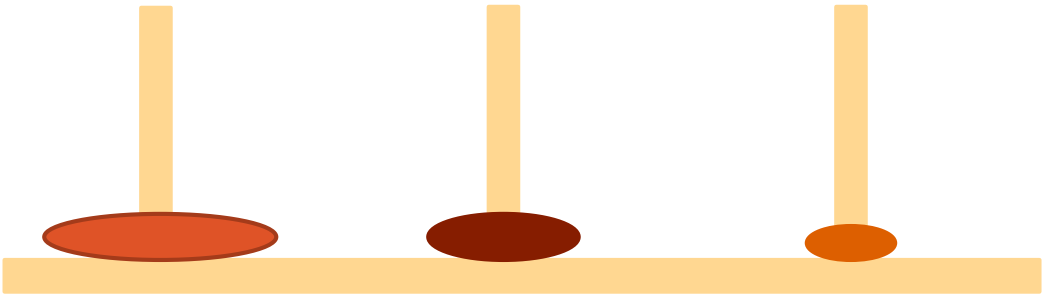


1-3	1-2	3-2	1-3	2-1	2-3	1-3

Example Tower of Hanoi

1-3	1-2	3-2	1-3	2-1	2-3	1-3

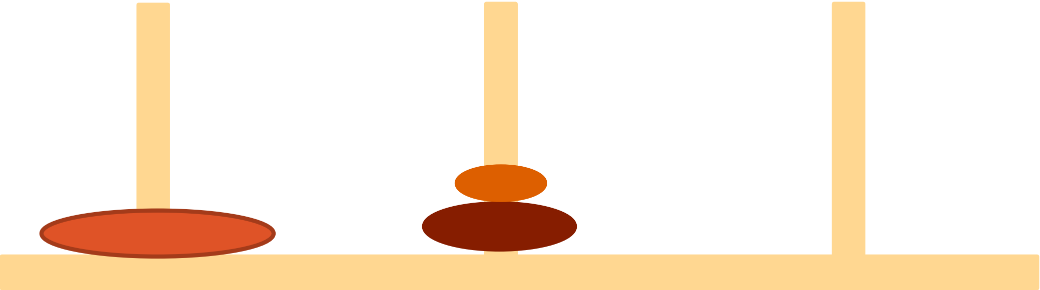
1-3



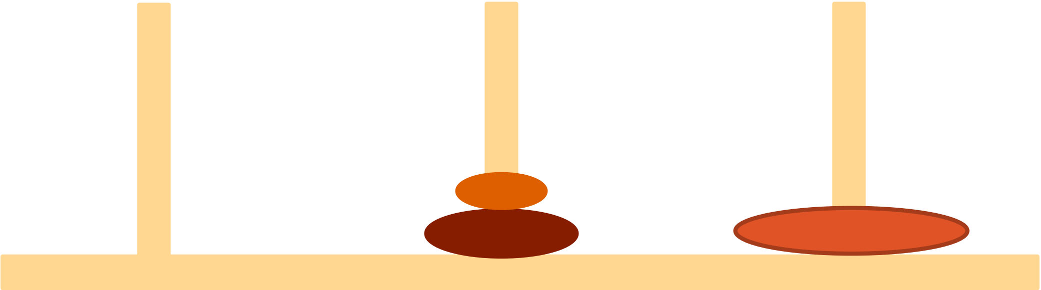
1-2

Example Tower of Hanoi

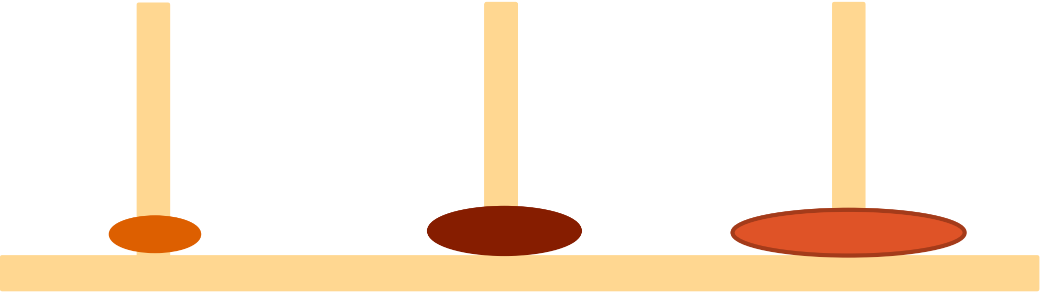
1-3	1-2	3-2	1-3	2-1	2-3	1-3



3-2



1-3



2-1

2-3

Example Tower of Hanoi

1-3	1-2	3-2	1-3	2-1	2-3	1-3

1-3

Run time stack tracing

Consider the function

```
int f(int x) {
 int y;
 if(x==0)
        return 1;
        else {
        y = 2 * f(x-1);
        return y + 1;
        }
    }
```