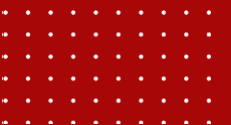
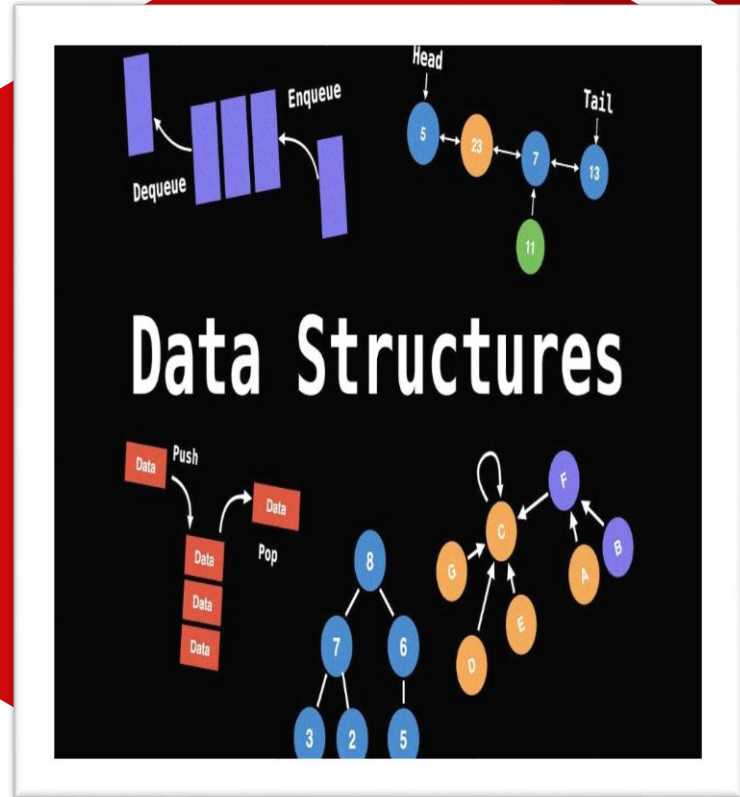


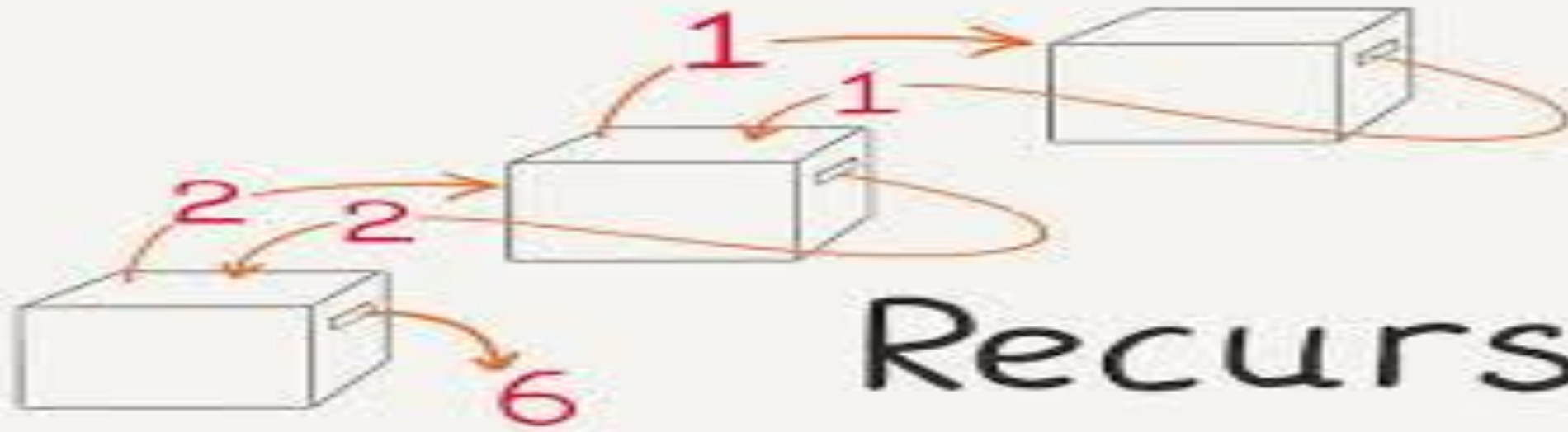
Algorithms and Data Structures

Recursion

S e s s i o n : D a y 2

Dr Kiran Waghmare
CDAC Mumbai



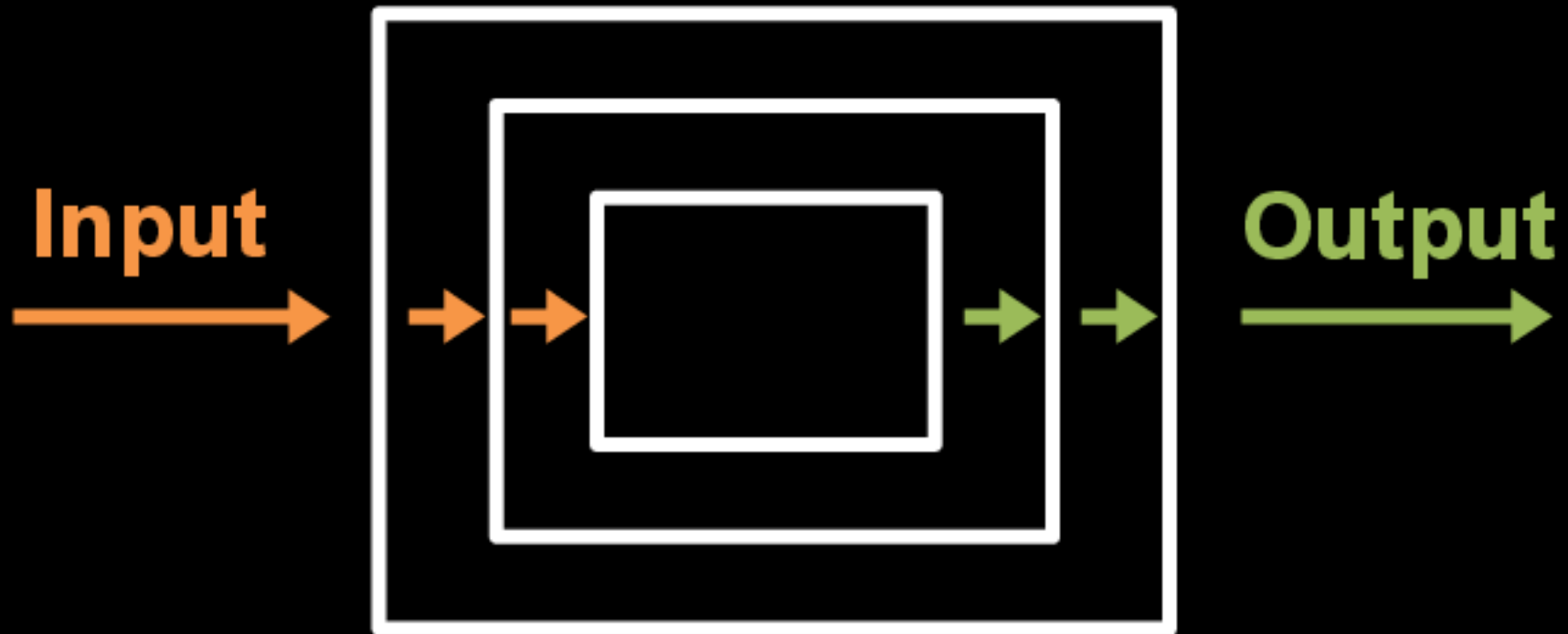


Recursion

Topics

1. Recursive definitions and Processes
2. Writing Recursive Programs
3. Efficiency in Recursion
4. Towers of Hanoi problem.

Recursion



How does Recursion works?

```
void recurse()
{
    ... ..
    recurse();
    ... ..
}

int main()
{
    ... ..
    recurse();
    ... ..
}
```

The diagram illustrates the flow of recursive calls. It shows two functions: `void recurse()` and `int main()`. Inside `recurse()`, there is a call to `recurse();`. Inside `main()`, there is a call to `recurse();`. Arrows indicate the sequence of calls: one arrow points from the `recurse();` line in `main()` to the opening curly brace of `recurse()`, and another arrow points from the `recurse();` line inside `recurse()` back to its own opening curly brace. The text "recursive call" is placed next to the arrow originating from `main()`.

Recursion

- Any function which calls itself directly or indirectly is called **Recursion** and the corresponding function is called as **recursive function**.
- A recursive method solves a problem by **calling a copy of itself** to work on a smaller problem.
- It is important to ensure that the **recursion terminates**.
- Each time the **function call itself** with a slightly simple version of the original problem.
- Using recursion, certain problems can be solved quite easily.
- E.g: Tower of Hanoi (TOH), Tree traversals, DFS of Graph etc.,

What is base condition in recursion?

- In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems.

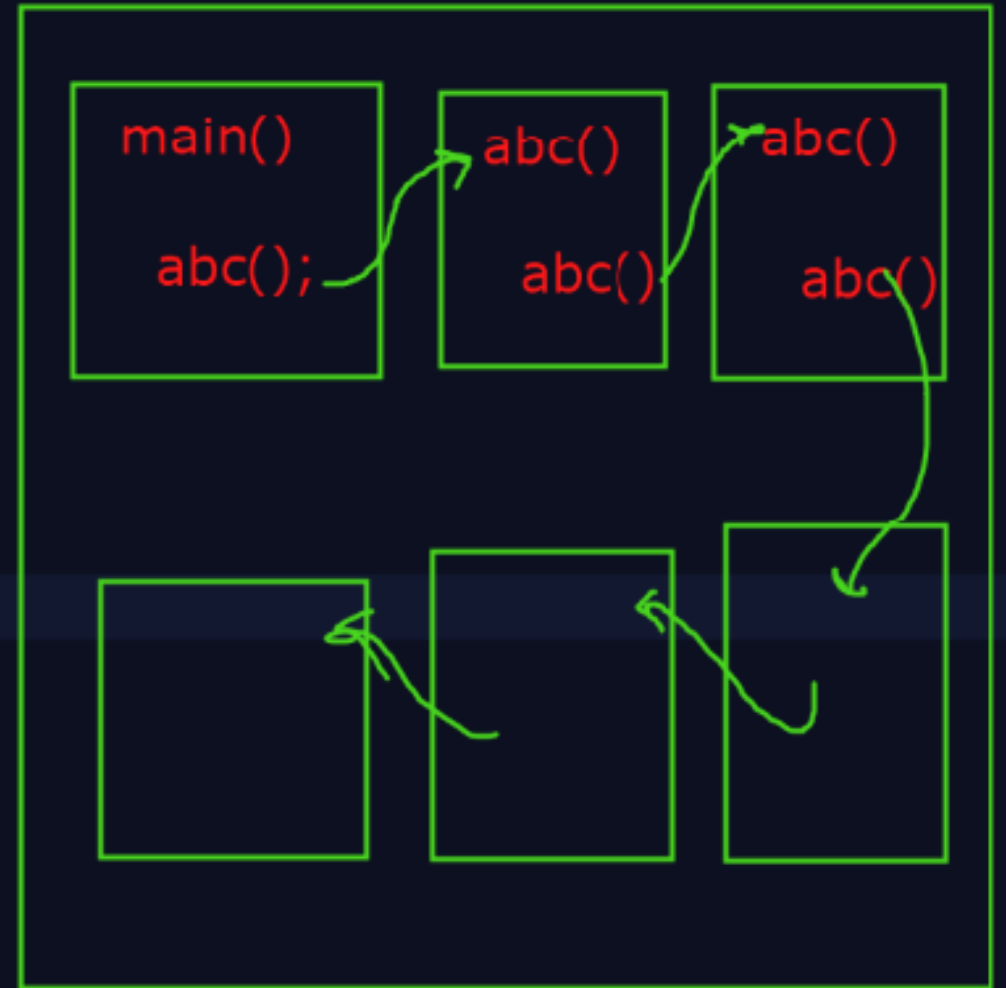
```
int fact(int n)
{
    if (n <= 1) // base case
        return 1;
    else
        return n*fact(n-1);
}
```

- In the above example, **base case for $n \leq 1$** is defined and larger value of number can be solved by converting to smaller one till base case is reached.

169
170 2 ways to implement the program:

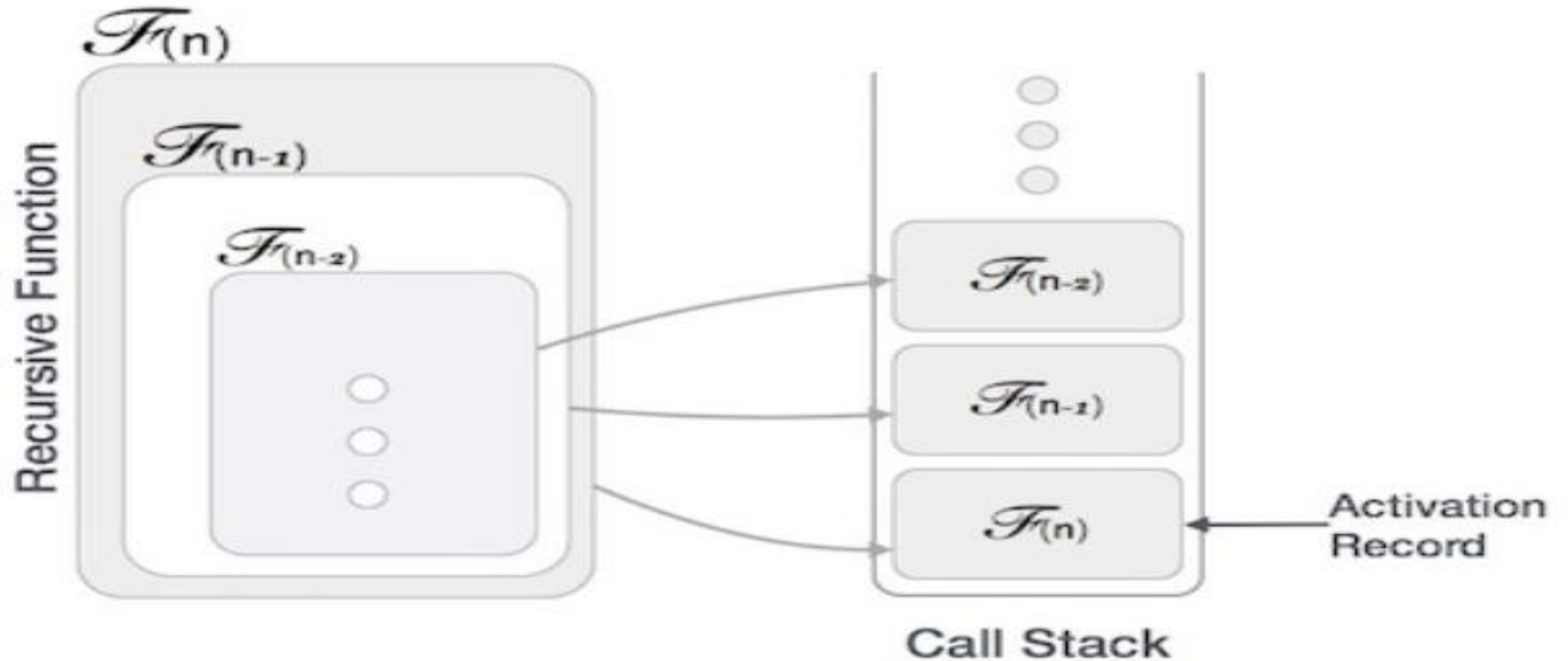
- 171
172 1.Iterative
173 2.Recursive

```
175 class A{  
176     abc() {  
177         abc();  
178     }  
179 }  
180  
181 main() {  
182  
183     abc();  
184 }  
185 }
```

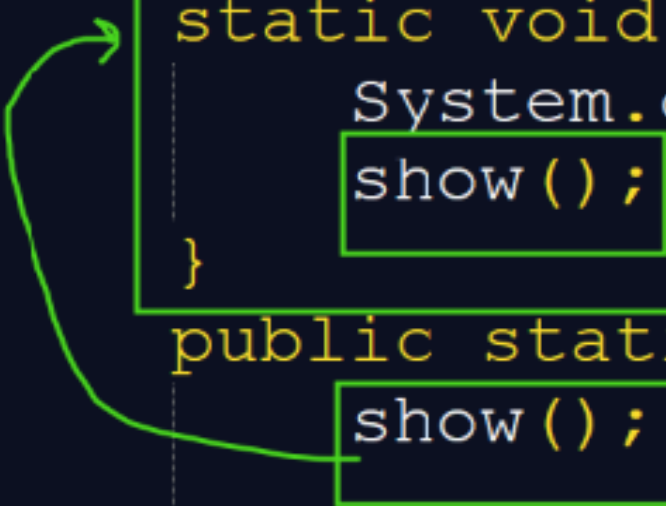


Stack overflow

How Data Structure Recursive function is implemented?




```
=class Recursion {  
=    static void show() {  
        System.out.println("Hi Girls....!!!");  
        show();  
    }  
=    public static void main(String[] args) {  
        show();  
    }  
}
```

A green arrow originates from the 'show()' call inside the 'main' method and points to the 'show()' method definition. Another green arrow originates from the 'show()' definition and points back to the 'show()' call inside 'main', illustrating the recursive call cycle.

```
//Recursion infinite loop
class Recursion1 {
    static int i=0;
    static void show() {
        i++;
        if(i<=5)           termination condition
        {
            System.out.println("Hi Girls!!!");
            show();
        }
    }
    public static void main(String[] args) {
        show();
    }
}
```

```
//Recursion infinite loop
```

```
class Recursion2 {
```

```
    static int show(int n){
```

```
        if (n==4) //base condition
```

```
        {
```

```
            return n;
```

```
        }
```

```
        //recursive condition
```

```
    else{
```

```
        return 2*show(n+1);
```

```
    }
```

```
}
```

```
public static void main(String[] args) {
```

```
    System.out.println(show(2));
```

```
}
```

```
}
```



show(2)

2*show(2+1)

2*(2*show(3+1))

Recursion Tree

stack call

~~show(4)~~

~~show(3)~~

~~show(2)~~

~~main()~~

4
8
16

```
//Recursion infinite loop
```

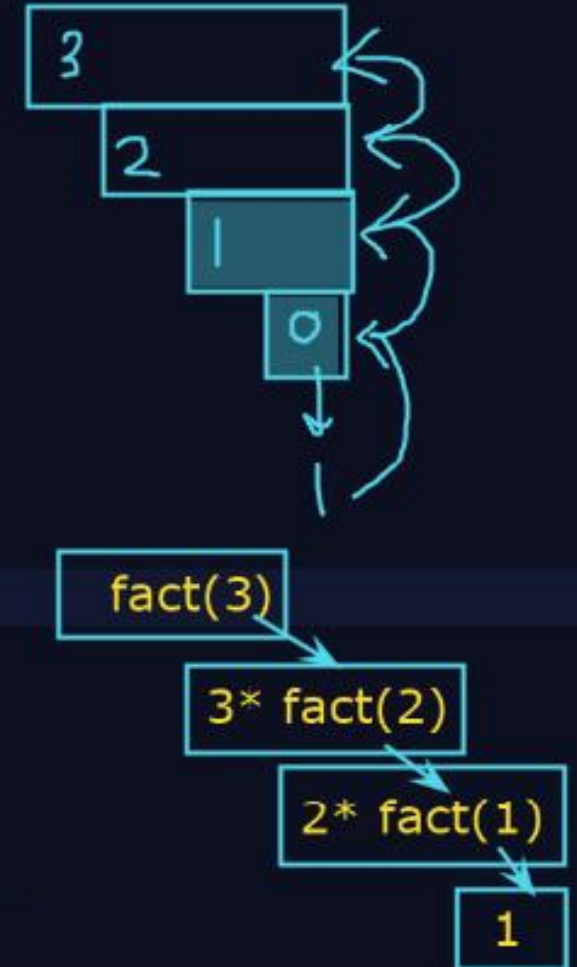
```
class Recursion3 {
```

```
    static int fact(int n){  
        if(n <= 1)  
            return 1;  
        else  
            return n*fact(n-1);  
    }
```

```
    public static void main(String[] args) {  
        System.out.println(fact(7));  
    }
```

```
}
```

$$\begin{aligned}\text{fact}(3) &= 3*2*1 \\ &= 1*2*3\end{aligned}$$



$$n * \text{fact}(n-1)$$

```
//Recursion infinite loop
```

```
class Recursion3 {
```

```
    static int fact(int n){  
        if(n <= 1)  
            return 1;  
        else  
            return n*fact(n-1);  
    }
```

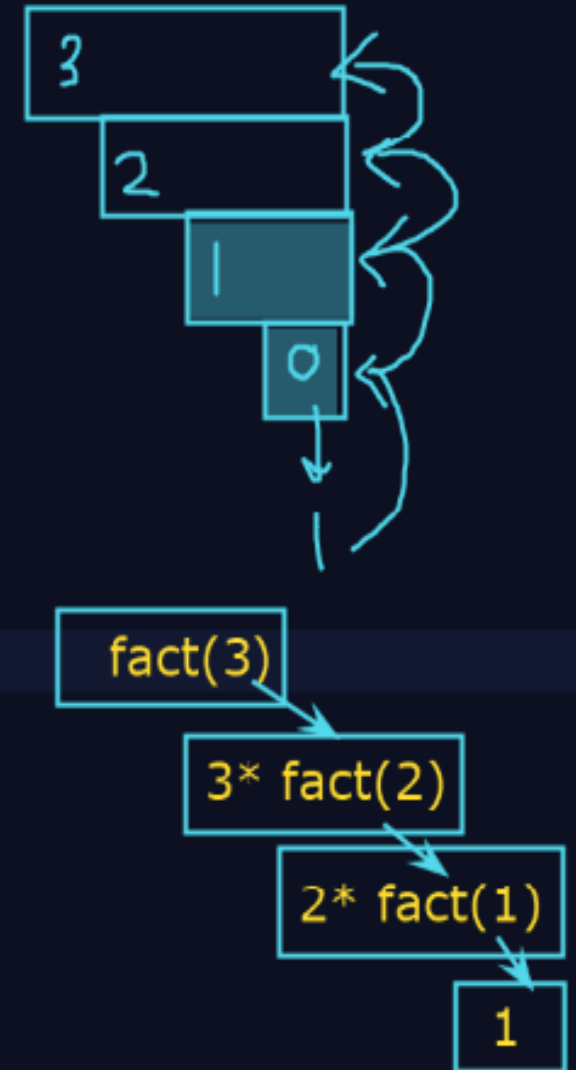
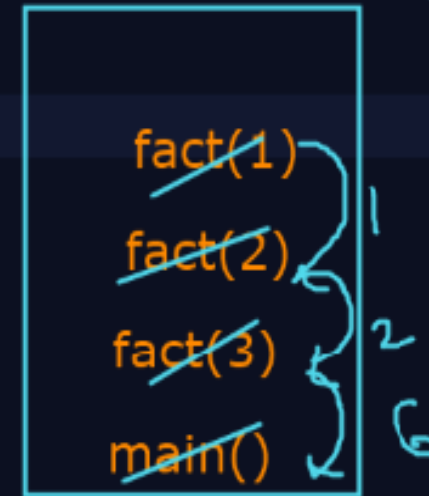
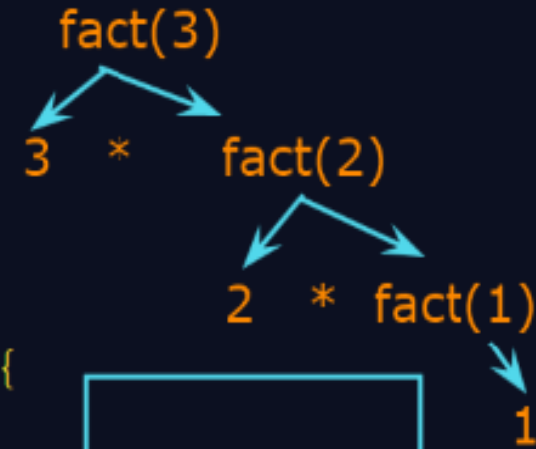
$$\text{fact}(3) = 3*2*1$$
$$= 1*2*3$$

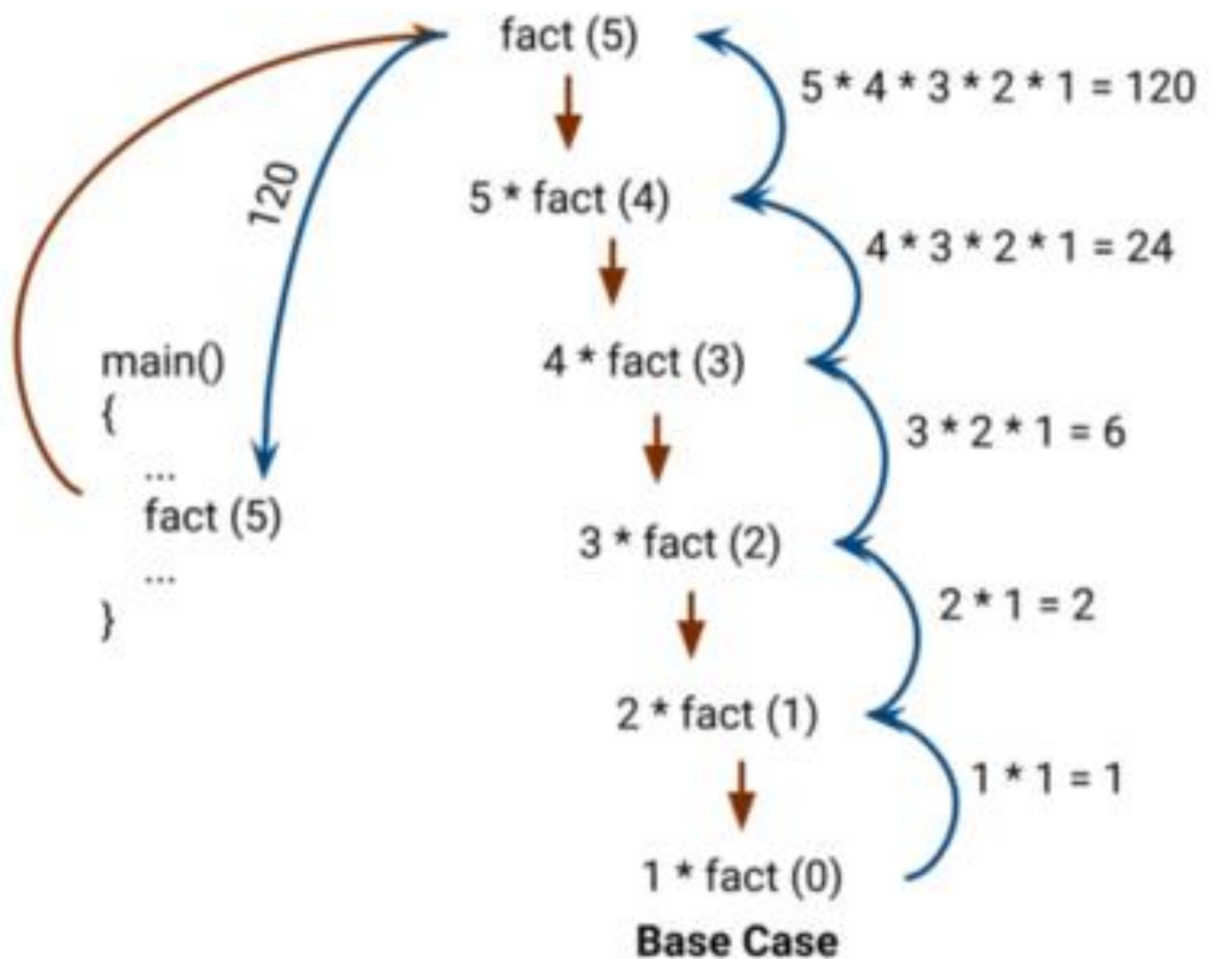
```
    public static void main(String[] args) {
```

```
        System.out.println(fact(7));
```

```
    }
```

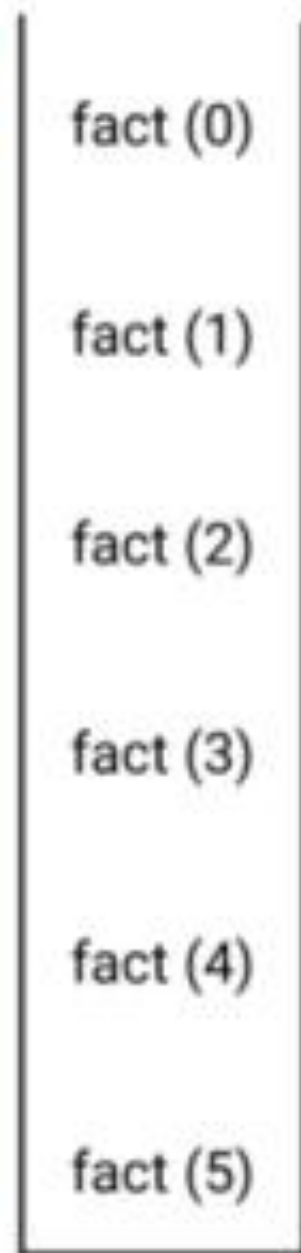
```
}
```





Recursion stop here and return the solution directly!

Order of execution of function call



Order in which functions
are returning values

Recursion Call Stack

How a particular problem is solved using recursion?

- The idea is to represent a problem in terms of one or more smaller problems, and add one or more base conditions that stop the recursion.
-
- For example, we compute factorial n if we know factorial of $(n-1)$.
- The base case for factorial would be $n = 0$.
- We return 1 when $n = 0$.

0 1 1 2 3 5 8

Fibonacci Series

$$0 + 1 = 1$$

$$1 + 1 = 2$$

$$1 + 2 = 3$$

$$2 + 3 = 5$$

$$3 + 5 = 8$$




```
//Fibonacci series:
```

```
f1=0, f2=1
```

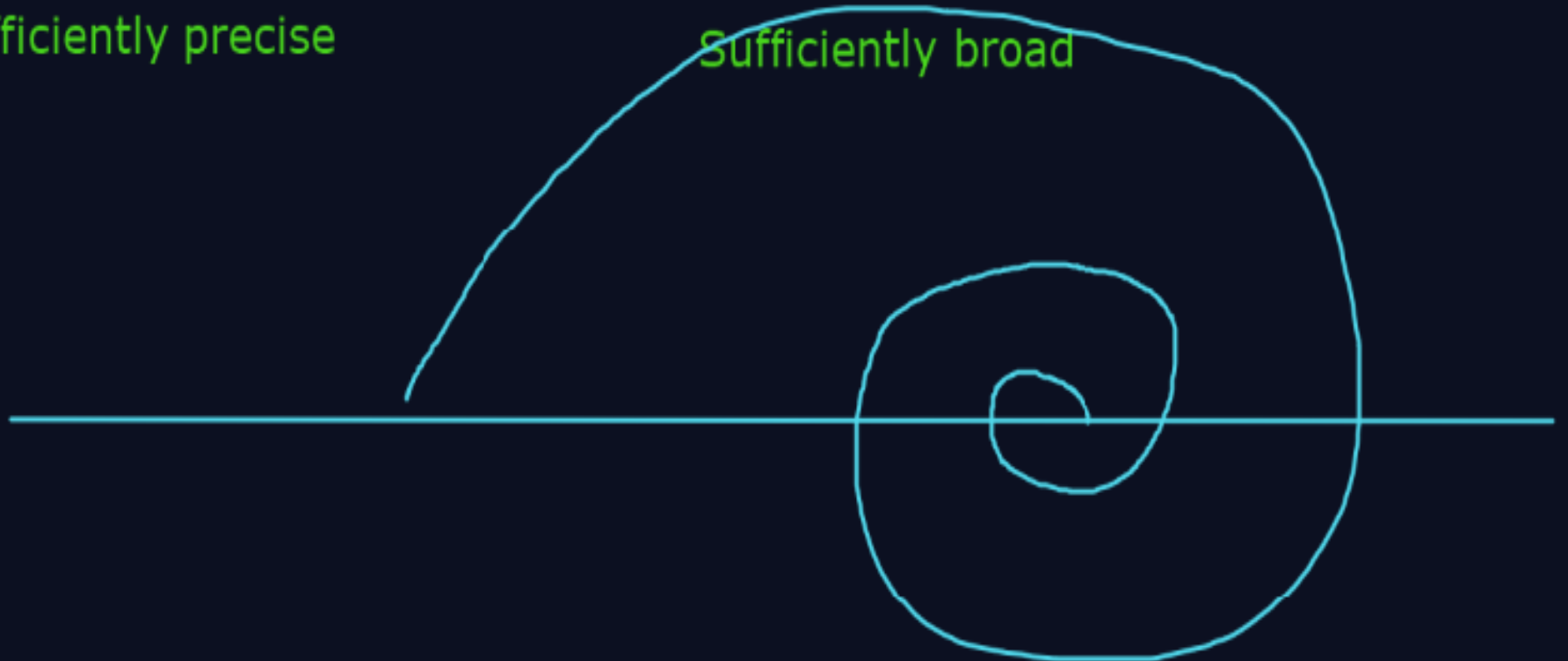
```
series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...
```

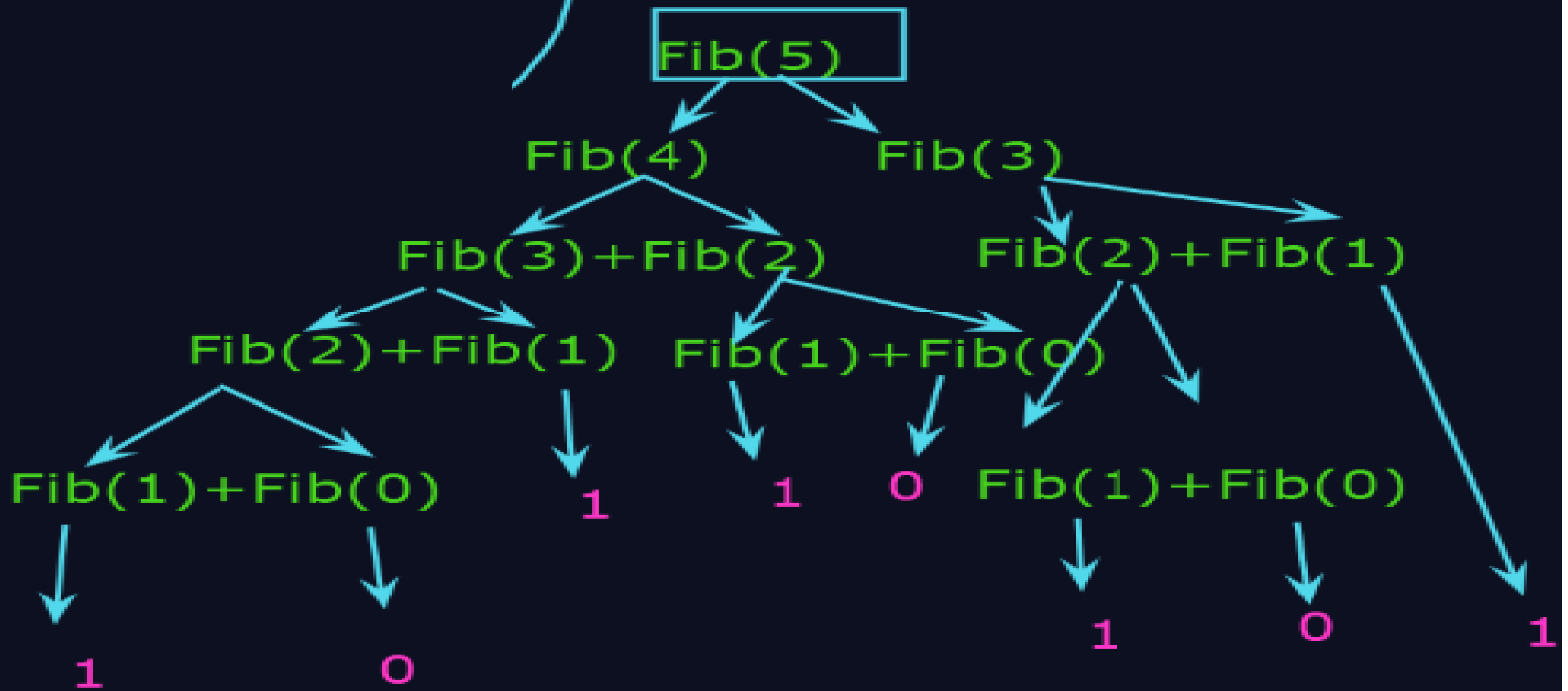
small difference

Large differences

Sufficiently precise

Sufficiently broad





Fibonacci gives teams precision *and* breadth in estimates



Sufficiently precise

Sufficiently broad

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89

Small items

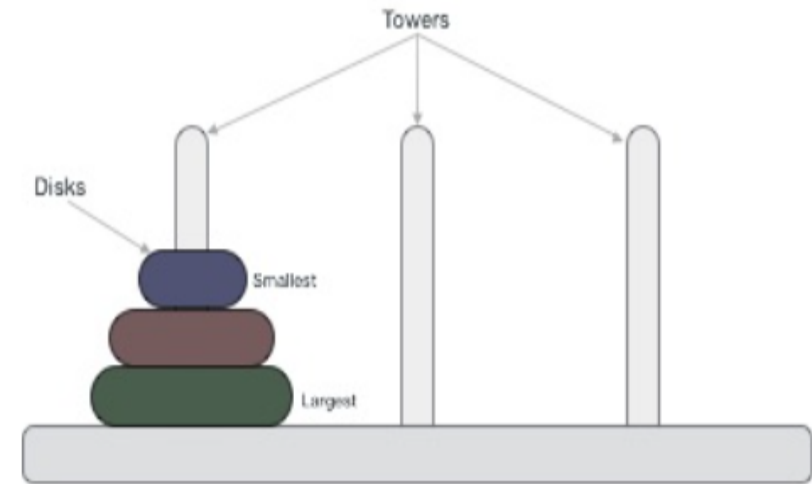
Large items

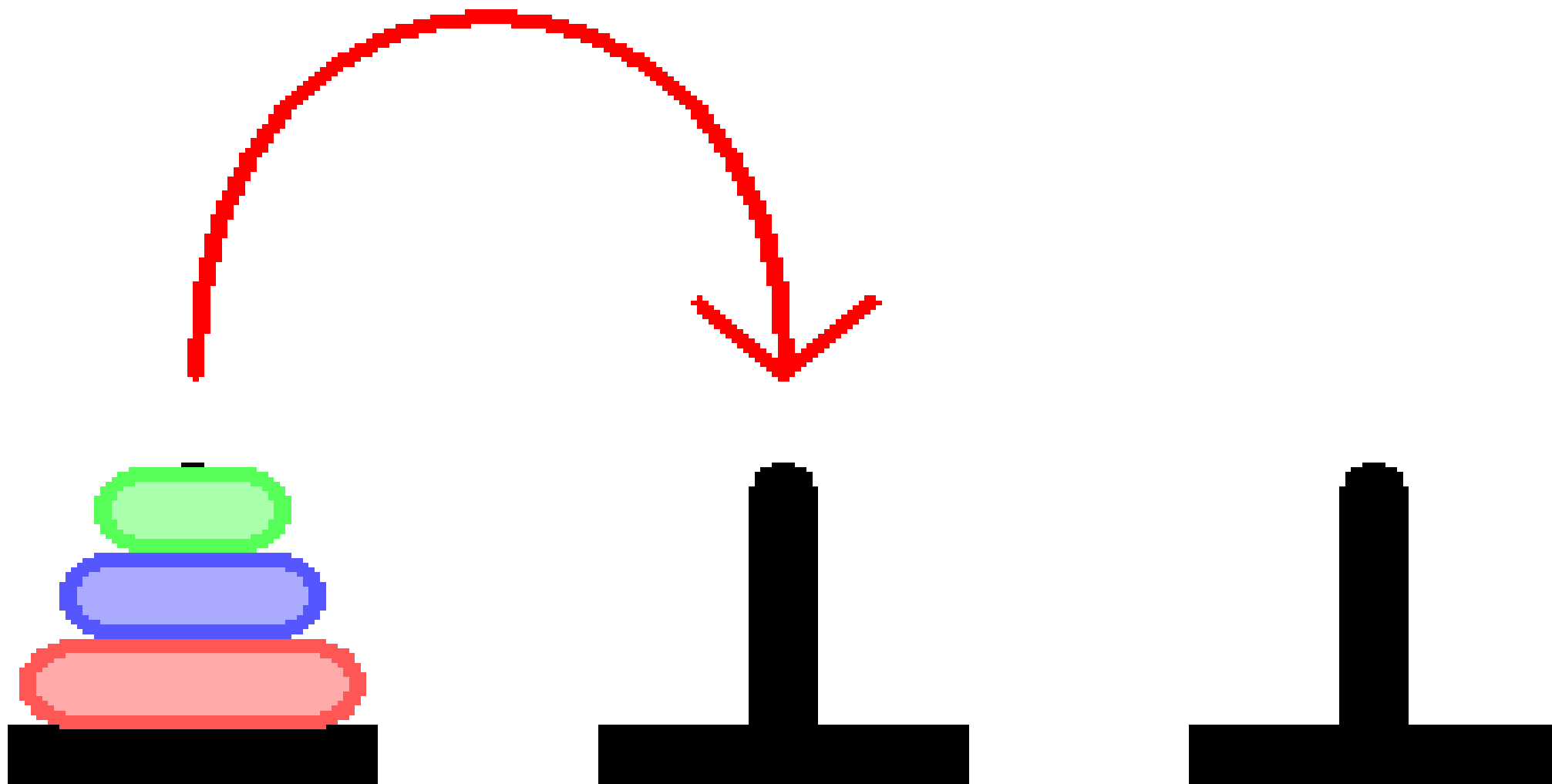
Why Algorithms?

- Fibonacci numbers
 - Compute first N Fibonacci numbers using iteration.
 - ... using recursion.
- Write the code.
- Try for N=5, 10, 20, 50, 100
- What do you see? Why does this happen?

What is Tower of Hanoi?

- A mathematical puzzle consisting of three towers and more than one ring is known as Tower of Hanoi.
- Tower of Hanoi
- The rings are of different sizes and are stacked in ascending order, i.e., the smaller one sits over the larger one. In some of the puzzles, the number of rings may increase, but the count of the tower remains the same.





What are the rules to be followed by Tower of Hanoi?

- **The Tower of Hanoi puzzle is solved by moving all the disks to another tower by not violating the sequence of the arrangements.**

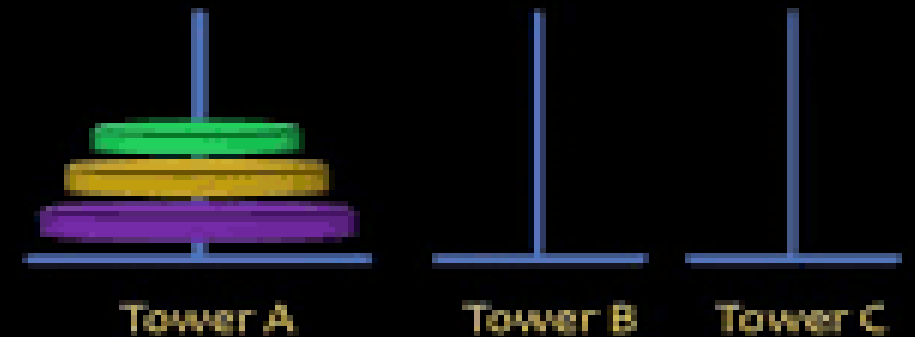
The rules to be followed by the Tower of Hanoi are -

1. Only one disk can be moved among the towers at any given time.
2. Only the "top" disk can be removed.
3. No large disk can sit over a small disk.

Algorithm 1: Recursive algorithm for solving Towers of Hanoi

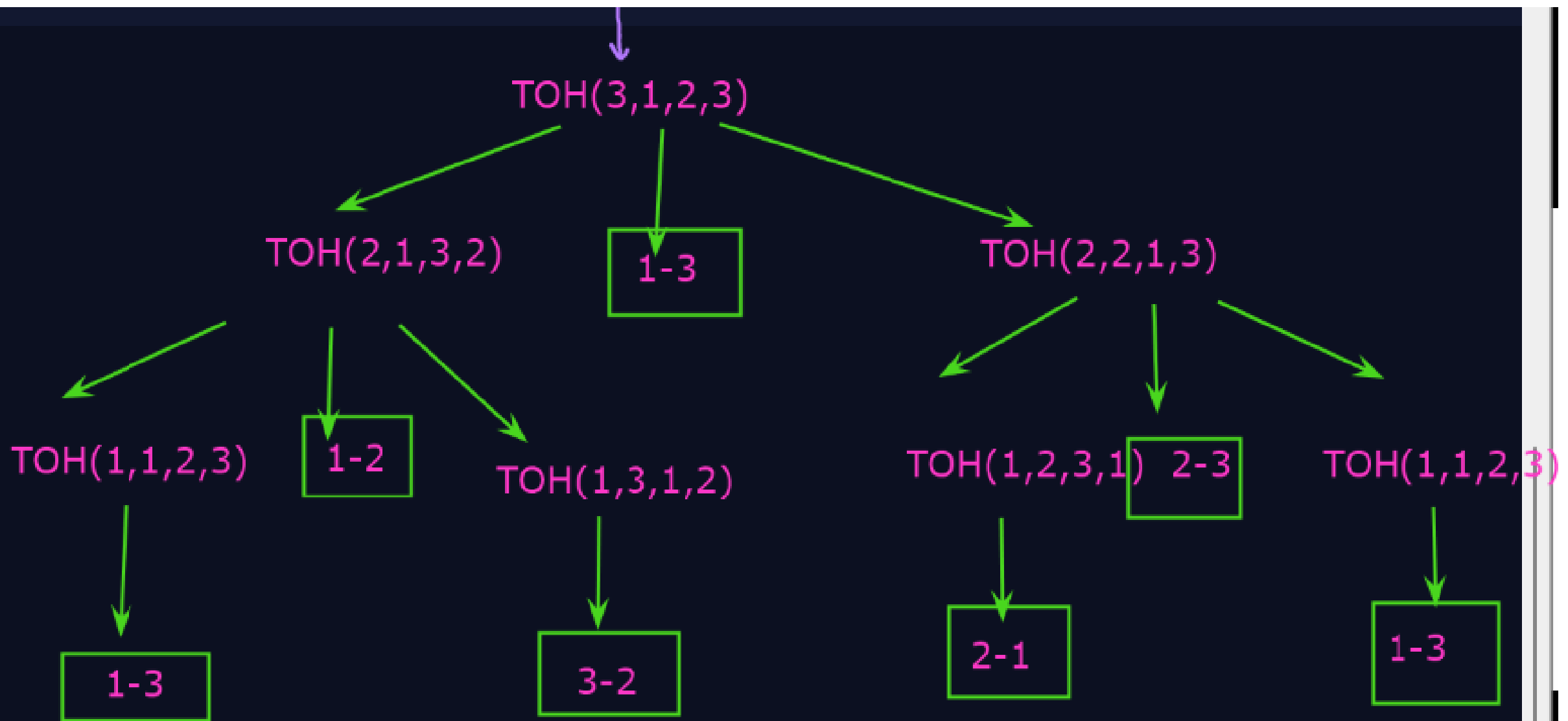
```
1 function recursiveHanoi( $n, s, a, d$ )
2   if  $n == 1$  then
3     print( $s + " \text{ to } " + d$ );
4     return;
5   end
6   recursiveHanoi( $n - 1, s, d, a$ );
7   print( $s + " \text{ to } " + d$ );
8   recursiveHanoi( $n - 1, a, s, d$ );
9 end
```

What is time complexity
of Tower of Hanoi
Problem?



Time Complexity: $O(2^n)$


```
class Recursion6 {  
    static void toh(int n, char s, char inter, char d) {  
        if (n == 1) {  
            System.out.println("Disk from "+s+" to "+d);  
        }  
        else {  
            toh(n-1, s, d, inter);  
            System.out.println("Disk from "+s+" to "+d);  
            toh(n-1, inter, s, d);  
        }  
    }  
  
    public static void main(String[] args) {  
        int n=3;  
        toh(n, 'A', 'B', 'C');  
    }  
}
```



Home Work

- Implement Tower of Hanoi Program
- No of Disk=3
- No of Disk=5
- No of Disk= n

Assignment 1

1. Print a series of numbers with recursive Java methods
2. Sum a series of numbers with Java recursion
3. Calculate a factorial in Java with recursion
4. Print the Fibonacci series with Java and recursion
5. A recursive Java palindrome checker

Recursion Types

Recursion is of two types based on when the call is made to the recursive method.

1) Tail Recursion

When the call to the recursive method is the last statement executed inside the recursive method, it is called

In tail recursion, the recursive call statement is usually executed along with the return statement of the method.

Tail Recursion: If a recursive function calling itself and that recursive call is the last statement in the function then it's known as **Tail Recursion**. After that call the recursive function performs nothing. The function has to process or perform any operation at the time of calling and it does nothing at returning time.

2) Head Recursion

Head recursion is any recursive approach that is not a tail recursion. So even general recursion is ahead recursion.

Head Recursion: If a recursive function calling itself and that recursive call is the first statement in the function then it's known as **Head Recursion**. There's no statement, no operation before the call. The function doesn't have to process or perform any operation at the time of calling and all operations are done at returning time.