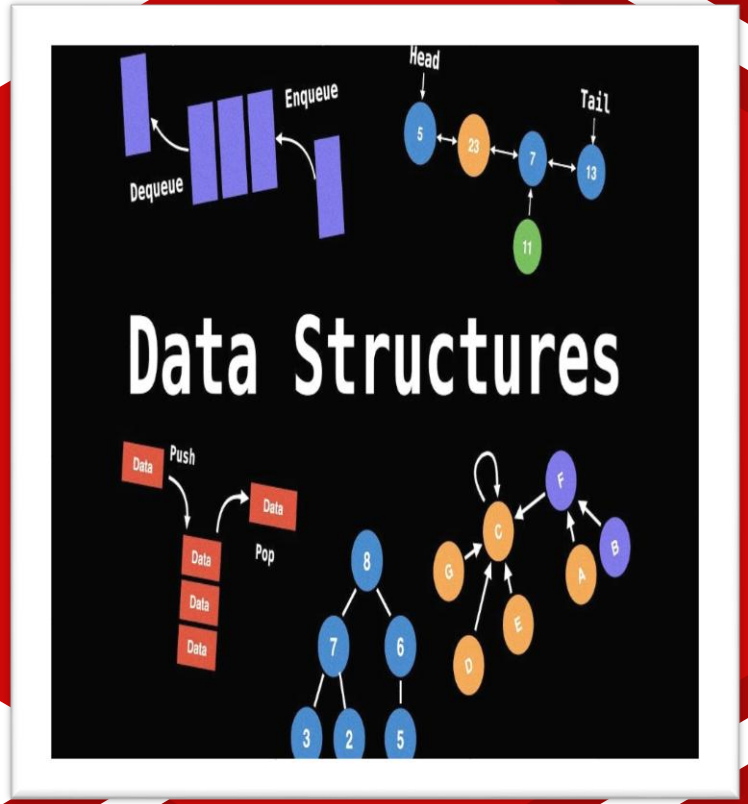
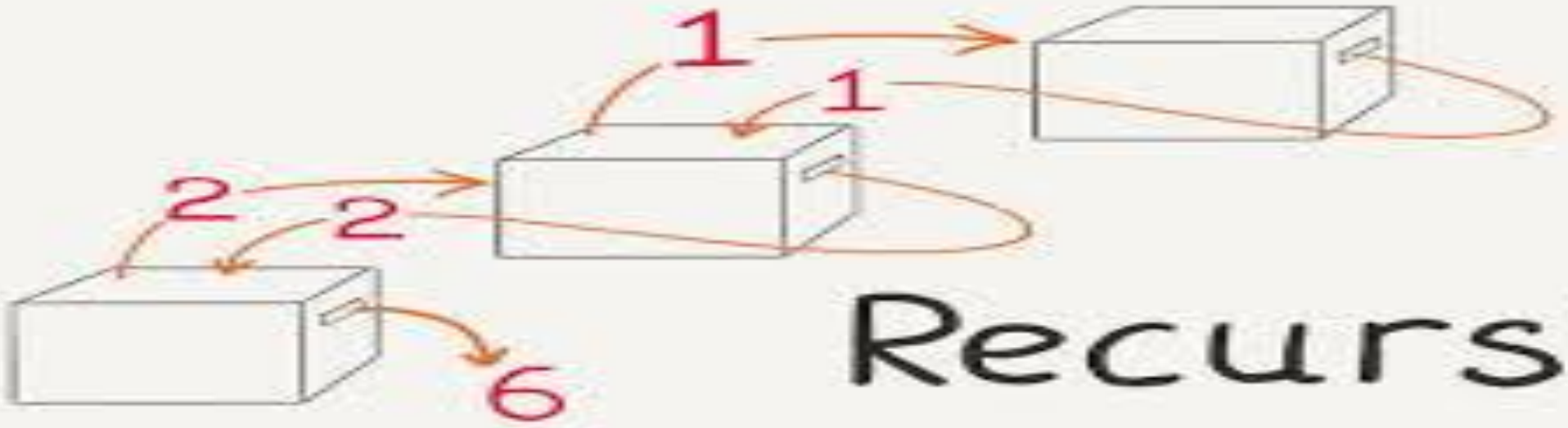
Algorithms and Data Structures



Recursion

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

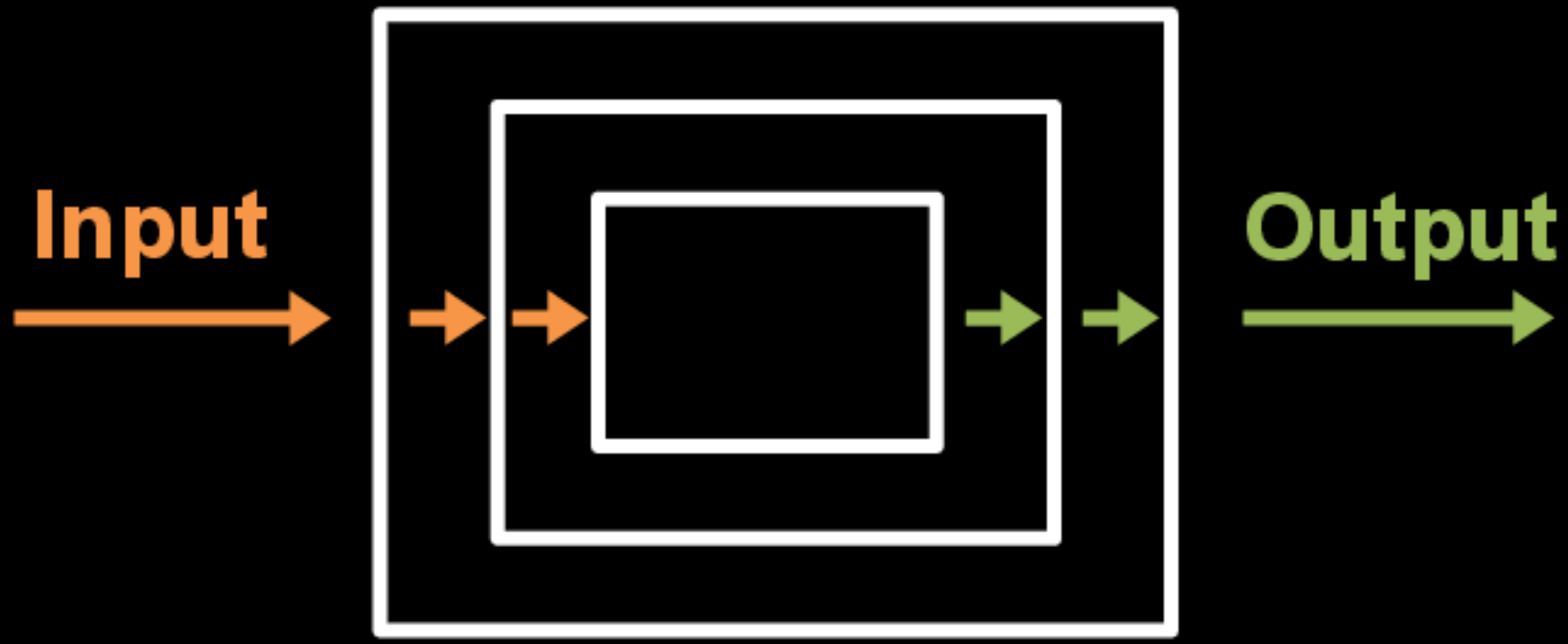
Dr Kiran Waghmare
CDAC Mumbai



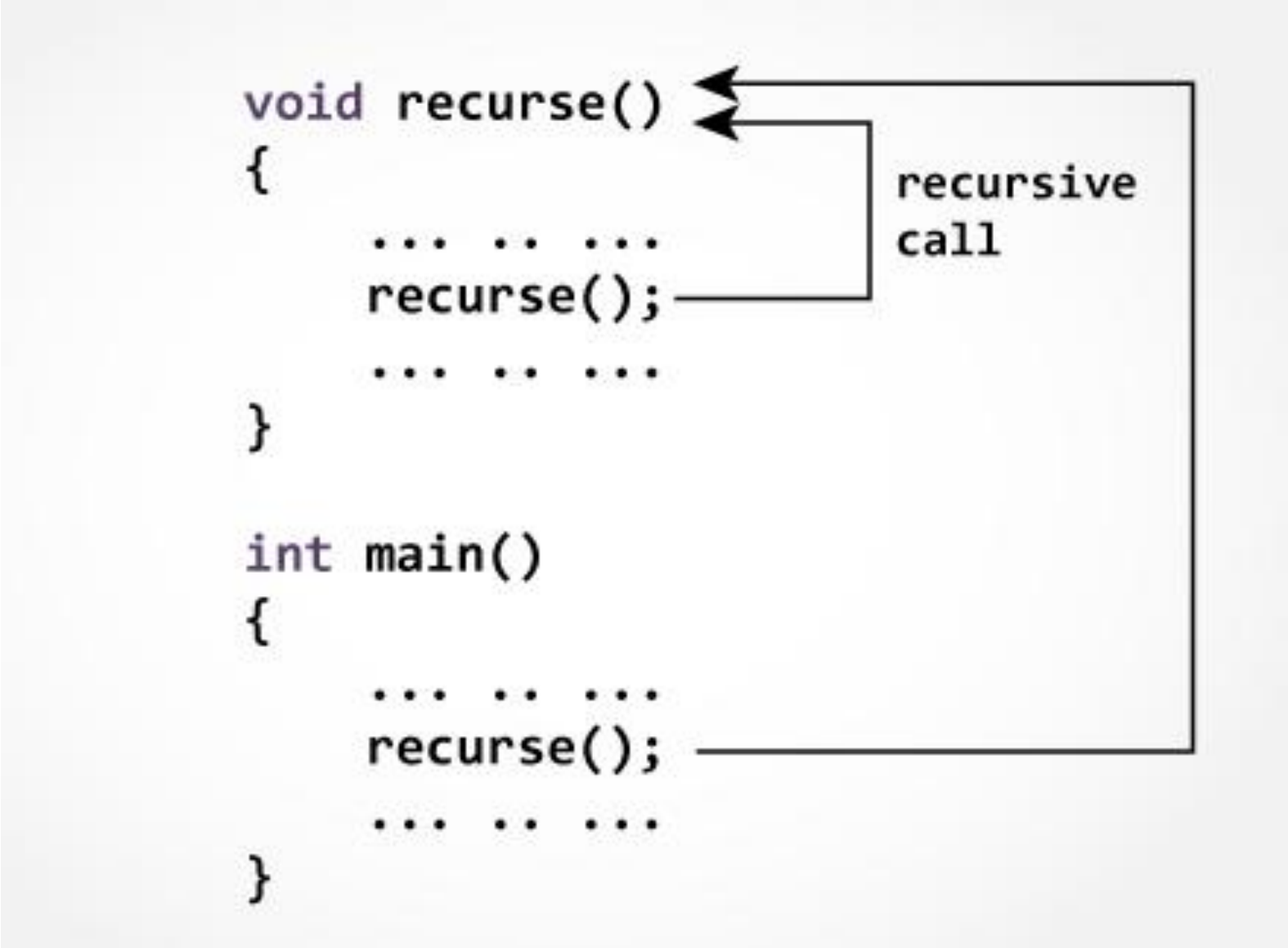
Topics

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

Recursion



How does Recursion works?



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.,

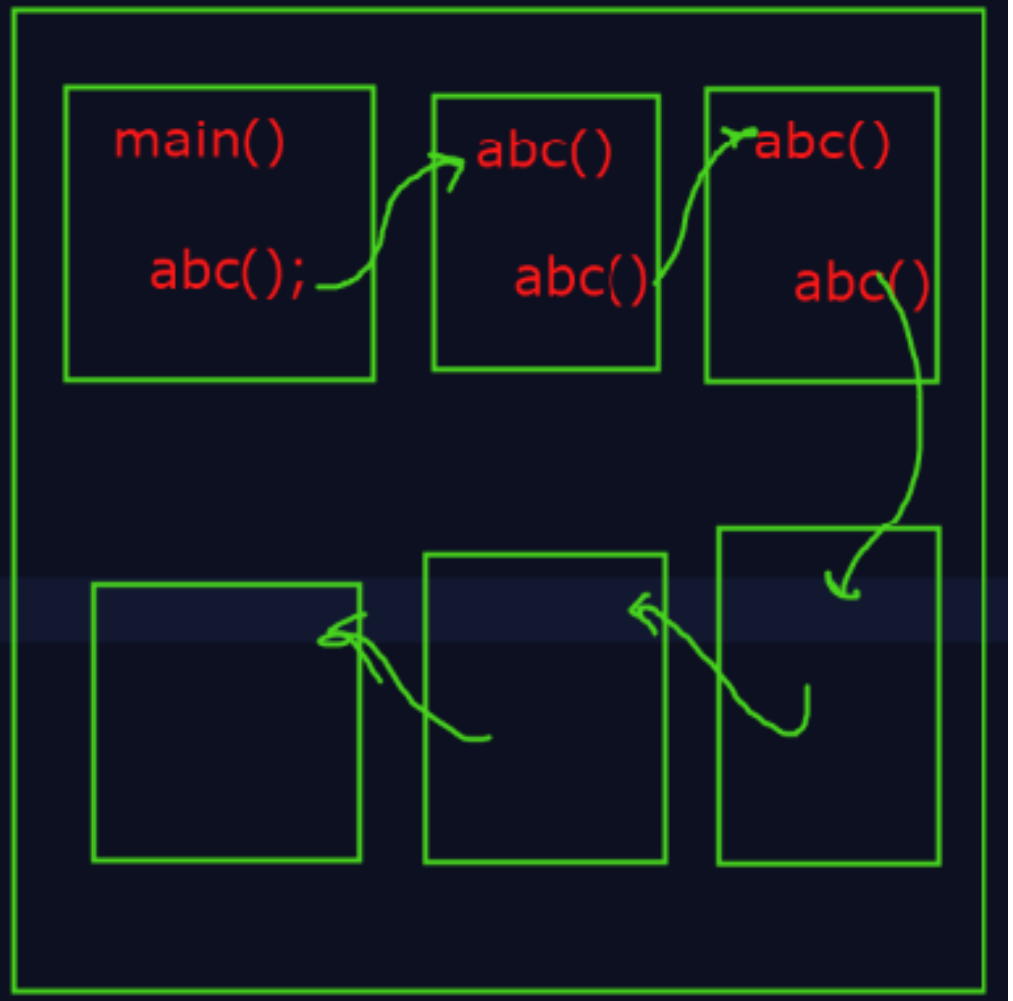
What is base condition in recursion?

- 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);
}
```

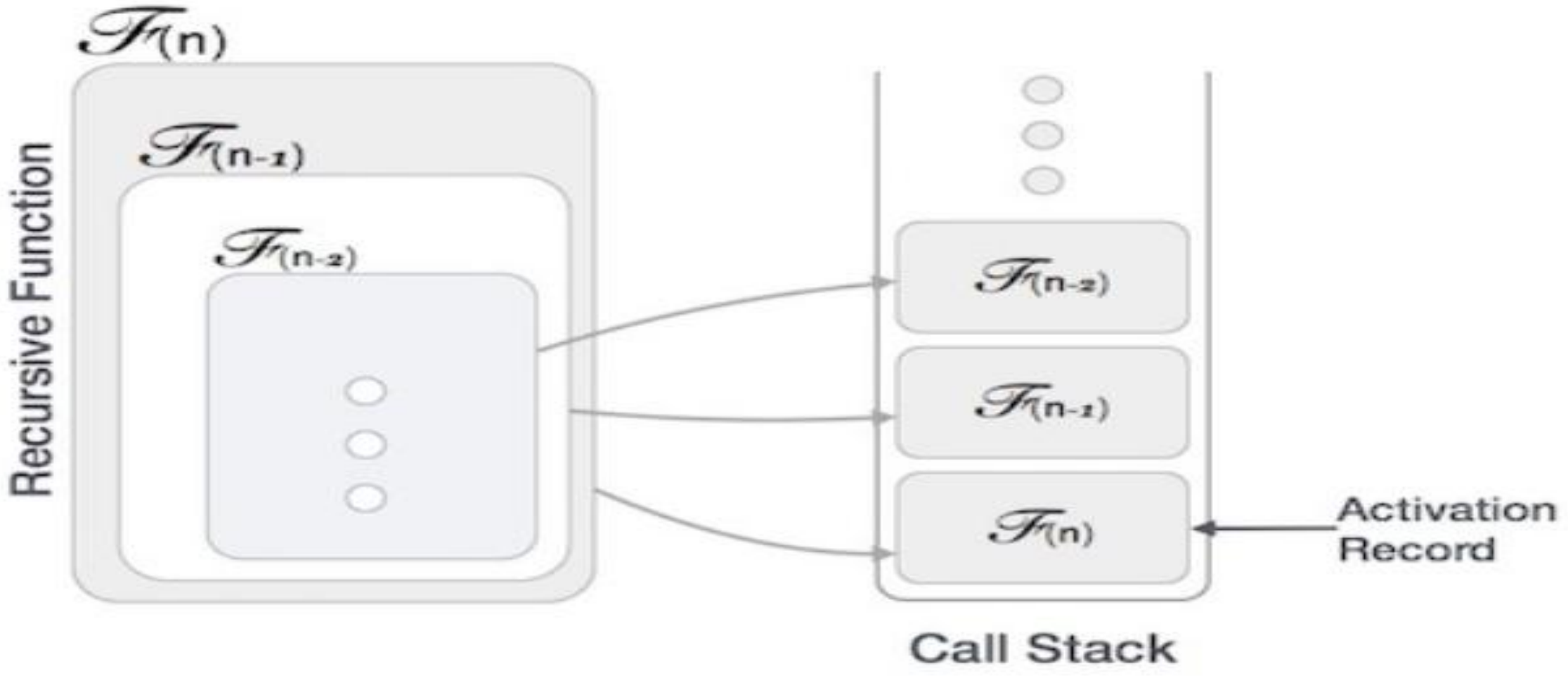
- In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

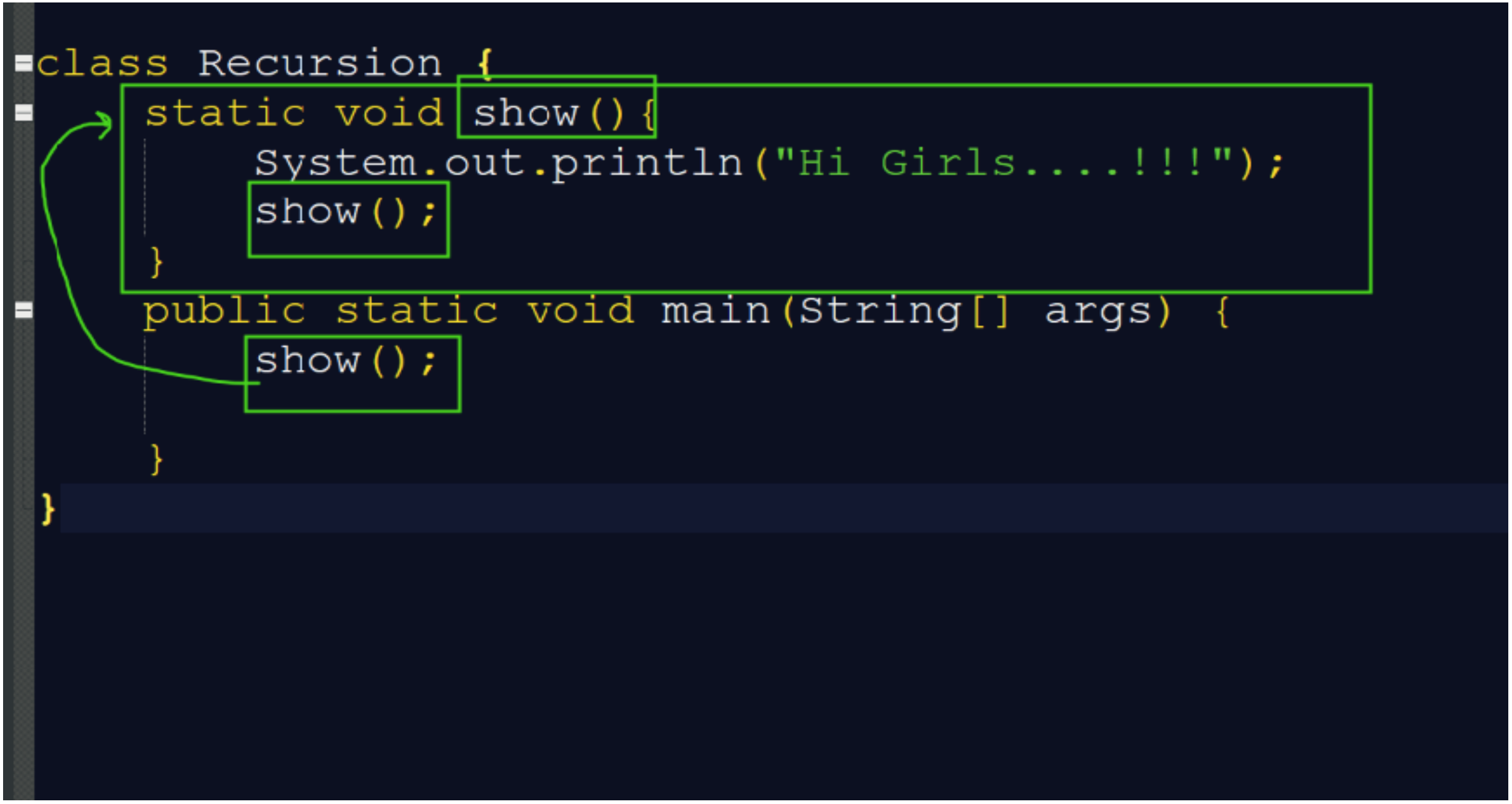
```
169
170  2 ways to implement the program:
171
172  1.Iterative
173  2.Recursive
174
175 class A{
176 abc(){
177
178      abc();
179  }
180
181 main(){
182
183      abc();
184  }
185  }
186
```



Stack overflow

How Data Structure Recursive function is implemented?

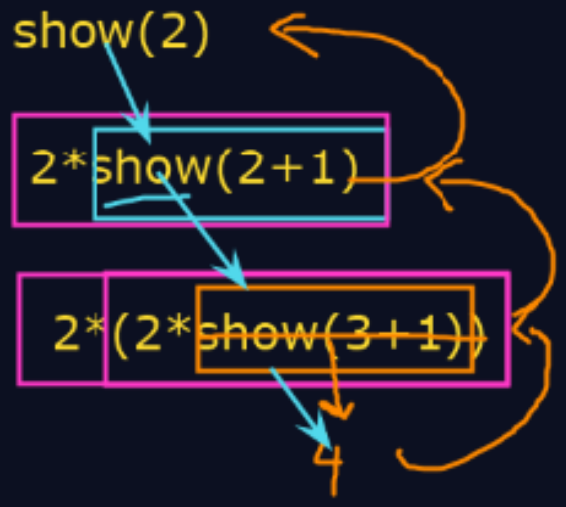



class Recursion {
static void show(){
System.out.println("Hi Girls....!!!");
show();
}
public static void main(String[] args) {
show();
}
}

```java
//Recursion infinite loop
class Recursion1 {
    static int i=0;
    static void show(){
        i++;
        if(i<=5)
        {
            System.out.println("Hi Girls!!!");
            show();

        }

    }
    public static void main(String[] args) {
        show();

    }
}
```

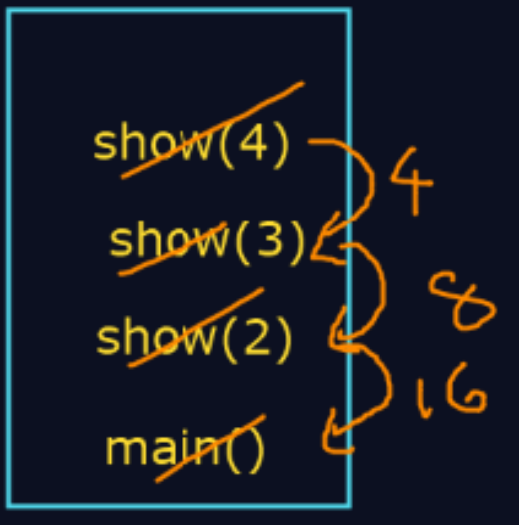
termination condition

```java
//Recursion infinite loop
class Recursion2 {

    static int show(int n){

        if(n==4)//base condition
        {
            return n;
        }
        //recusive condition
        else{
            return 2*show(n+1);

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

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



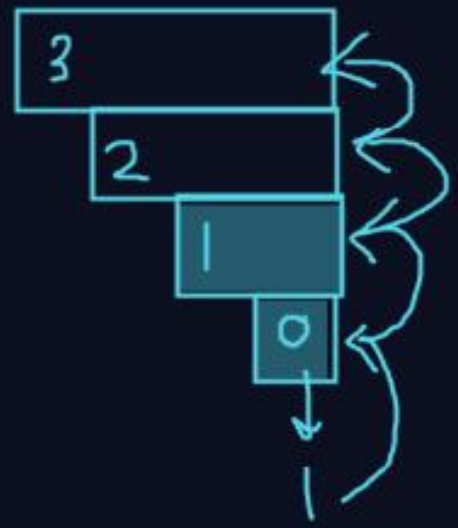
Recursion Tree

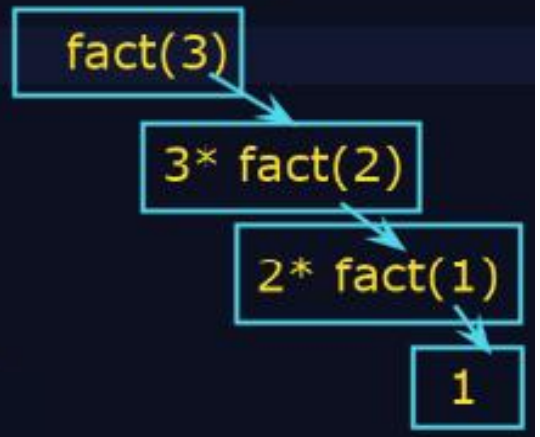
stack call



```java
//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));
    }
}
```

fact(3)= 3*2*1
=1*2*3

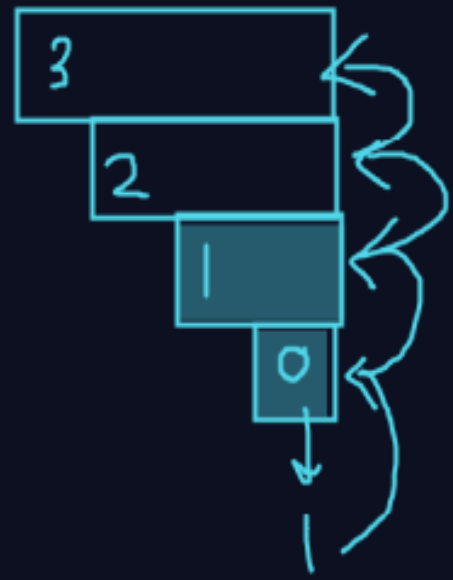


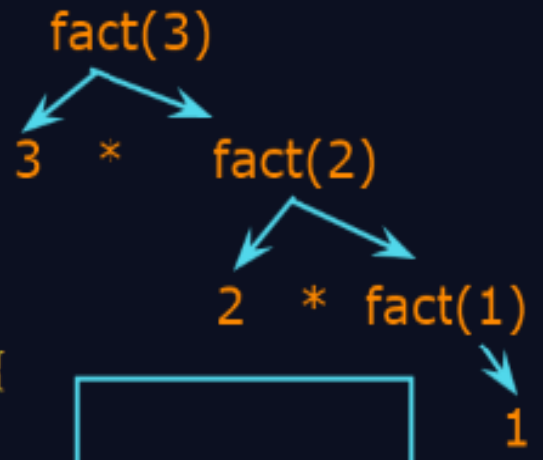


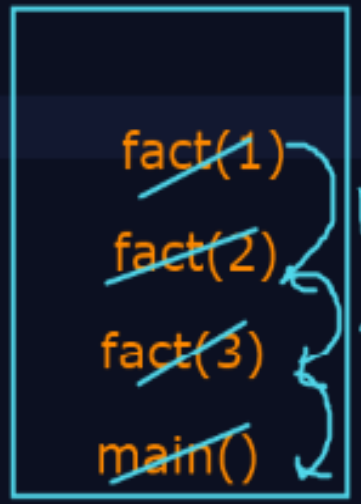
n*fact(n-1)

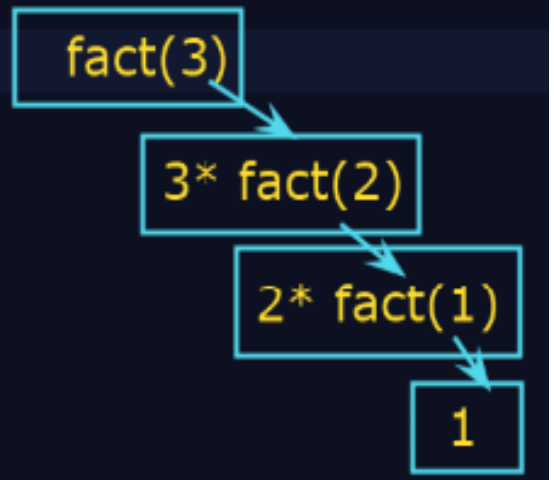
```java
//Recursion infinite loop
class Recursion3 {

    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));
    }
}
```

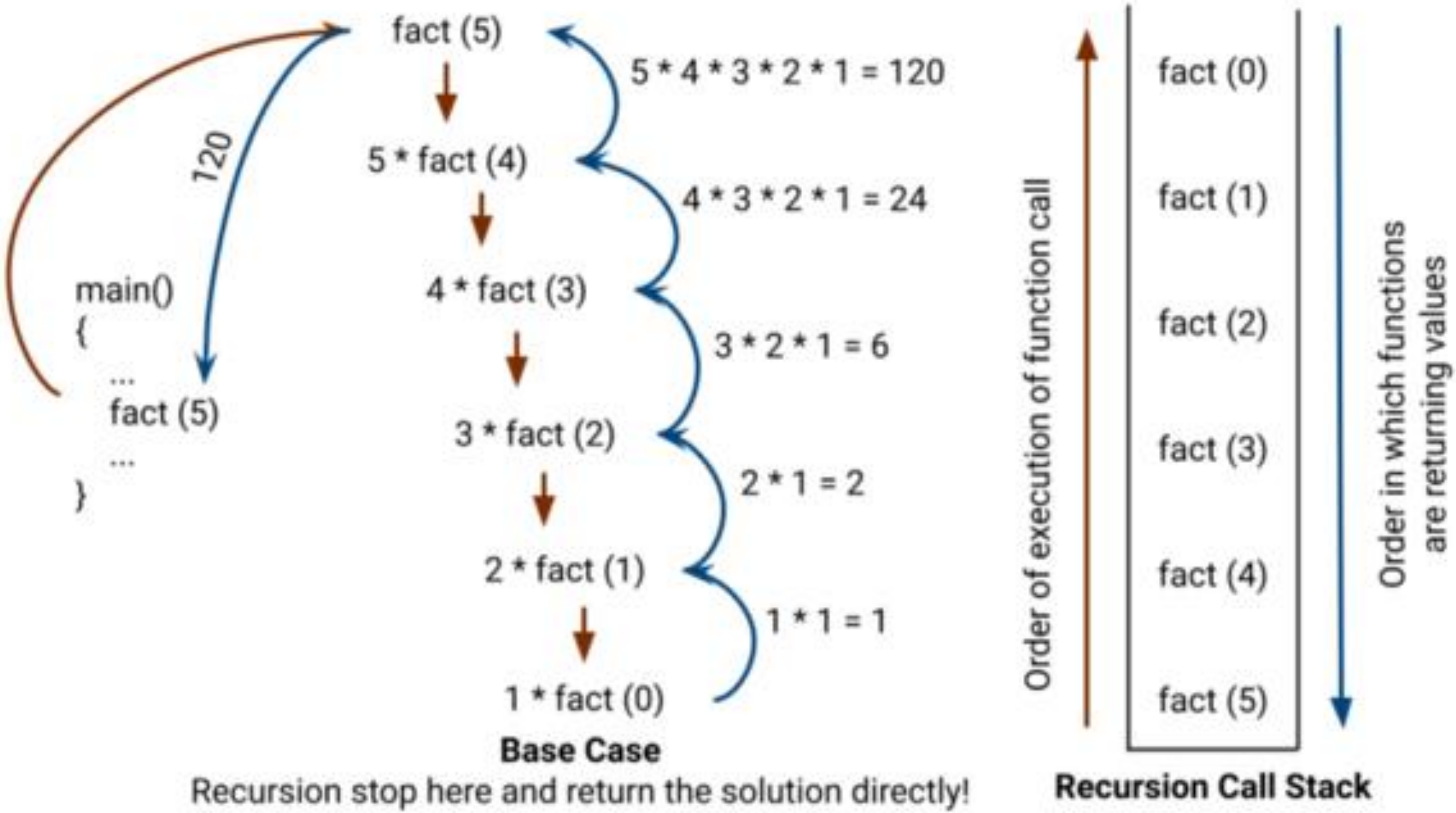
fact(3)= 3*2*1
=1*2*3










fact (5)
5 * 4 * 3 * 2 * 1 = 120
5 * fact (4)
4 * 3 * 2 * 1 = 24
4 * fact (3)
3 * 2 * 1 = 6
3 * fact (2)
2 * 1 = 2
2 * fact (1)
1 * 1 = 1
1 * fact (0)
120
main()
{
...
fact (5)
...
}
Base Case
Recursion stop here and return the solution directly!
Order of execution of function call
Order in which functions are returning values
fact (0)
fact (1)
fact (2)
fact (3)
fact (4)
fact (5)
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.

Fibonacci Series

0 1 1 2 3 5 8

0 + 1 = 1

1 + 1 = 2

1 + 2 = 3

2 + 3 = 5

3 + 5 = 8

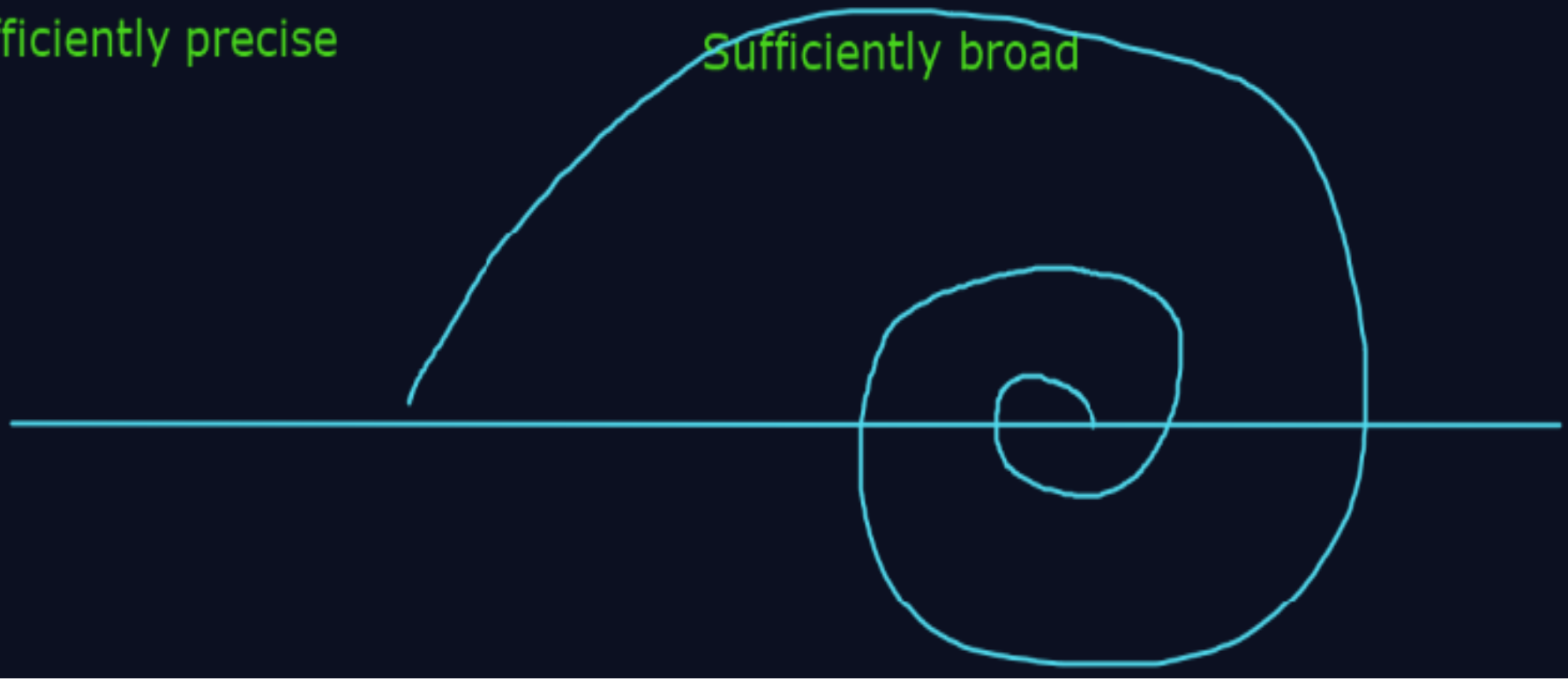
```
//Fibnacci series:

f1=0, f2=1

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

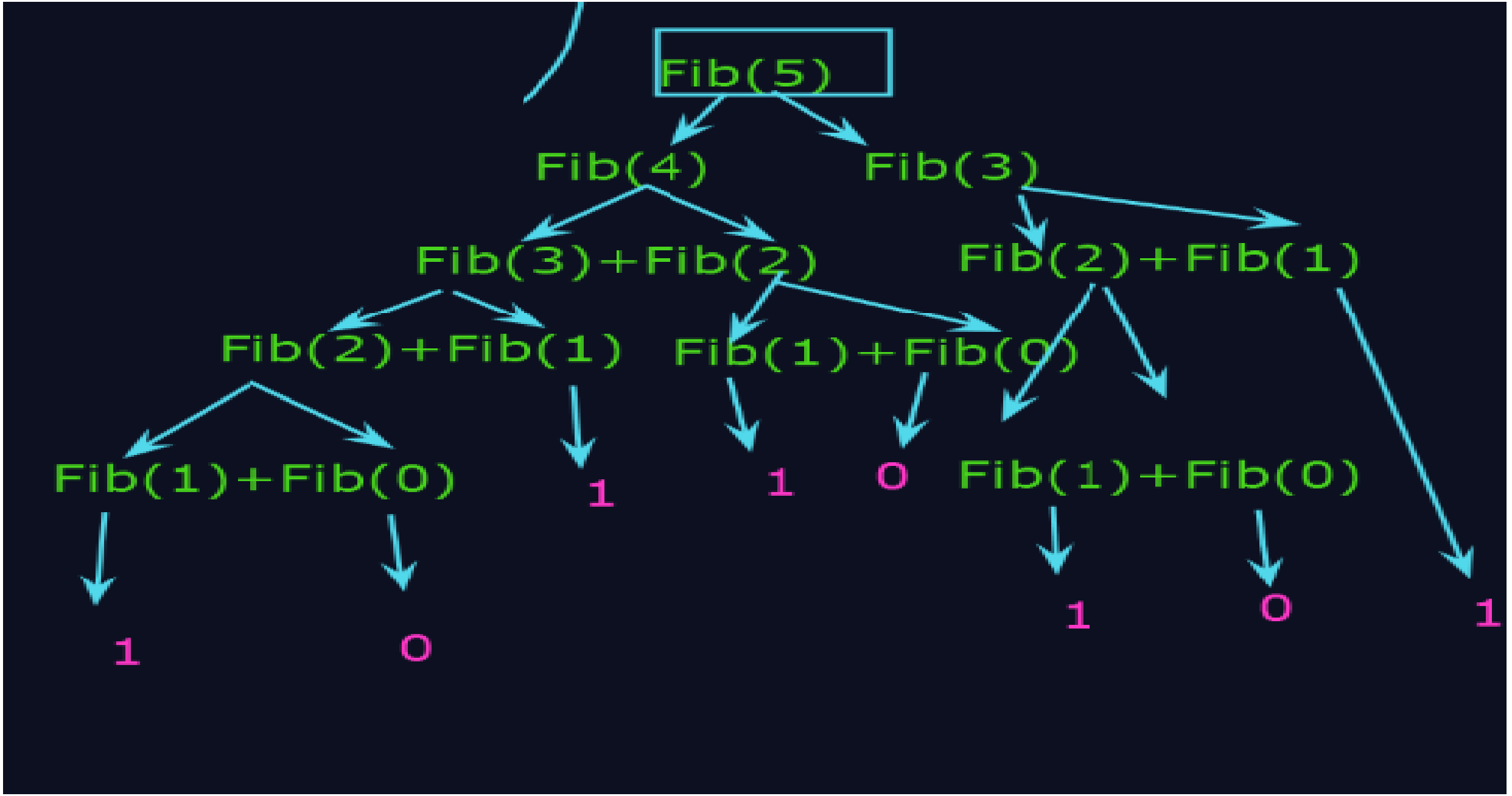
small difference

Sufficiently precise

Large differences

Sufficiently broad




Fib(5)
Fib(4)
Fib(3)
Fib(3)+Fib(2)
Fib(2)+Fib(1)
Fib(2)+Fib(1)
Fib(1)+Fib(0)
Fib(1)+Fib(0)
Fib(1)+Fib(0)
1
1
0
1
0
1
0
1

Fibonacci gives teams precision *and* breadth in estimates

parabol

Sufficiently precise

Sufficiently broad

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

Small items

Large items

Why Algorithms?

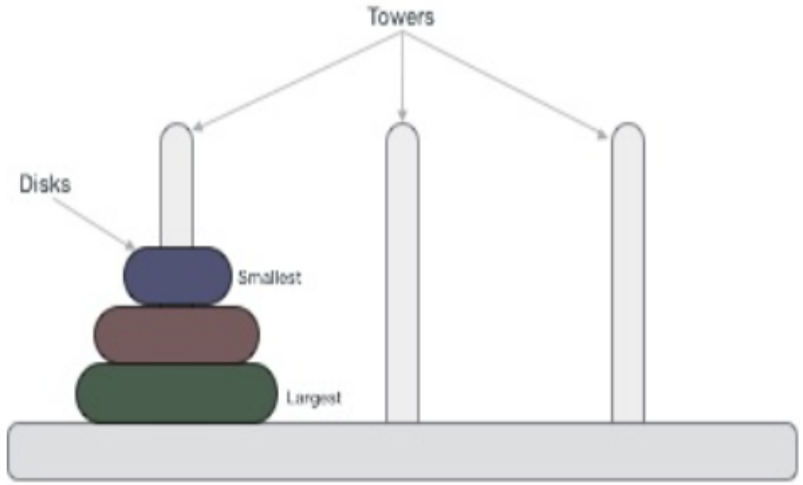
- Fibonacci numbers
  - Compute first N Fibonacci numbers using iteration.
  - … using recursion.
- Write the code.
- Try for N=5, 10, 20, 50, 100
- What do you see? Why does this happen?

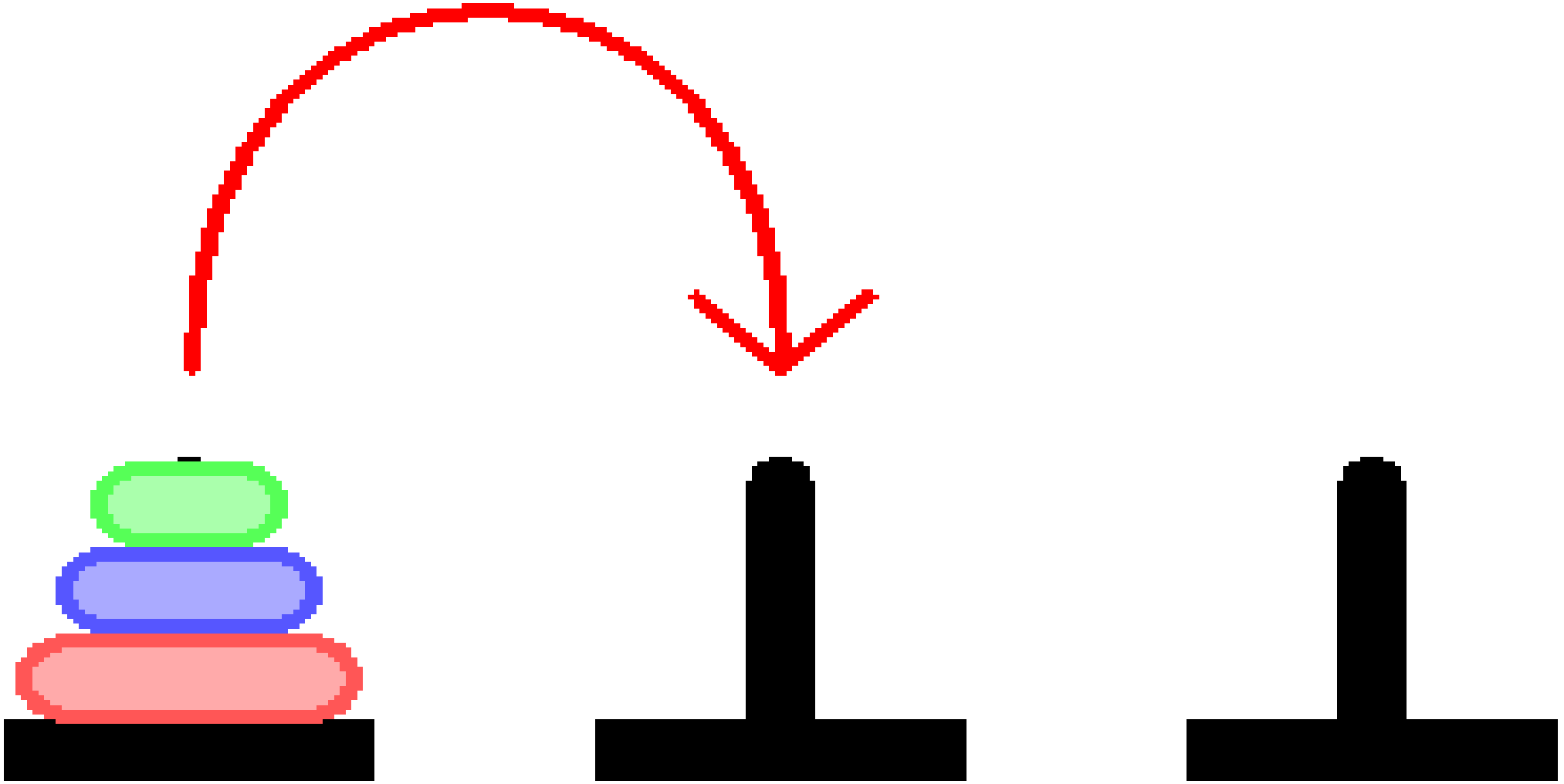
What is Tower of Hanoi?

- 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?

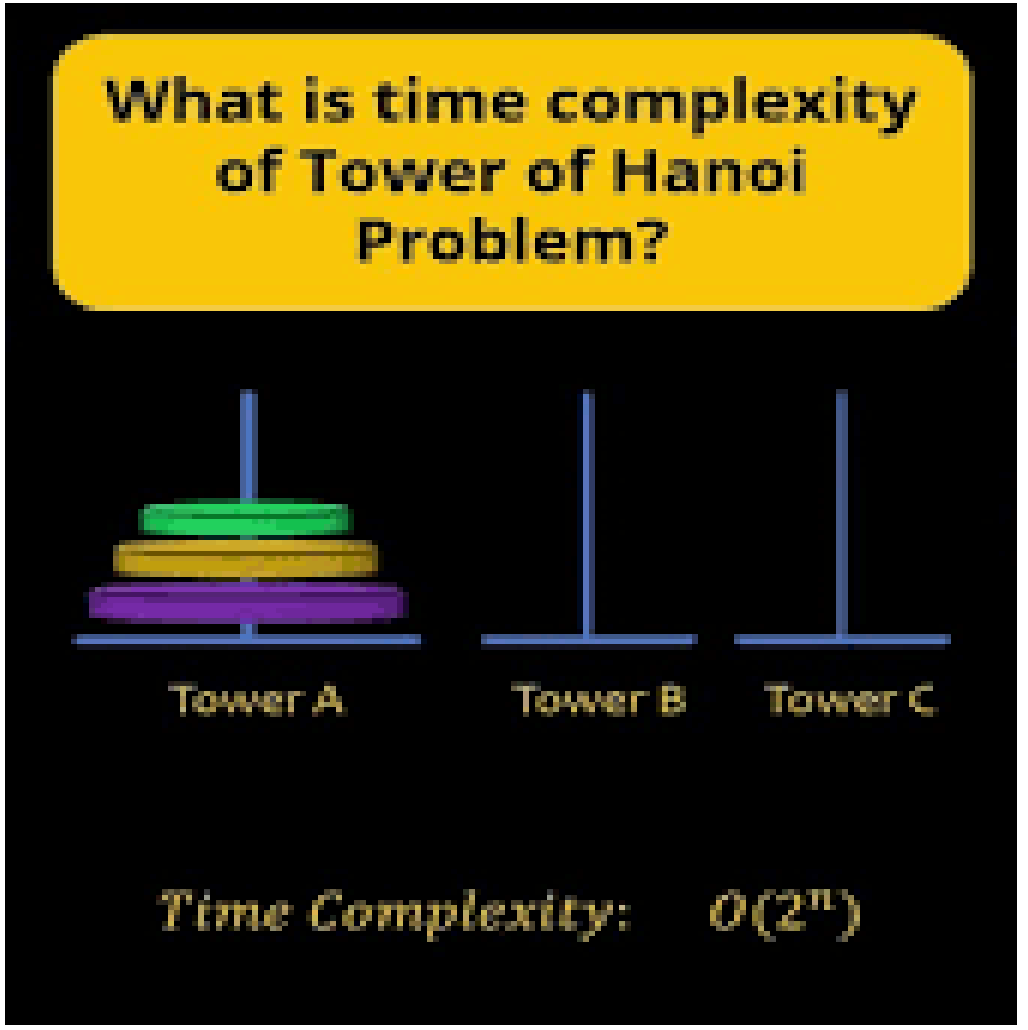
- **The Tower of Hanoi puzzle is solved by moving all the disks to another tower by not violating the sequence of the arrangements.**

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

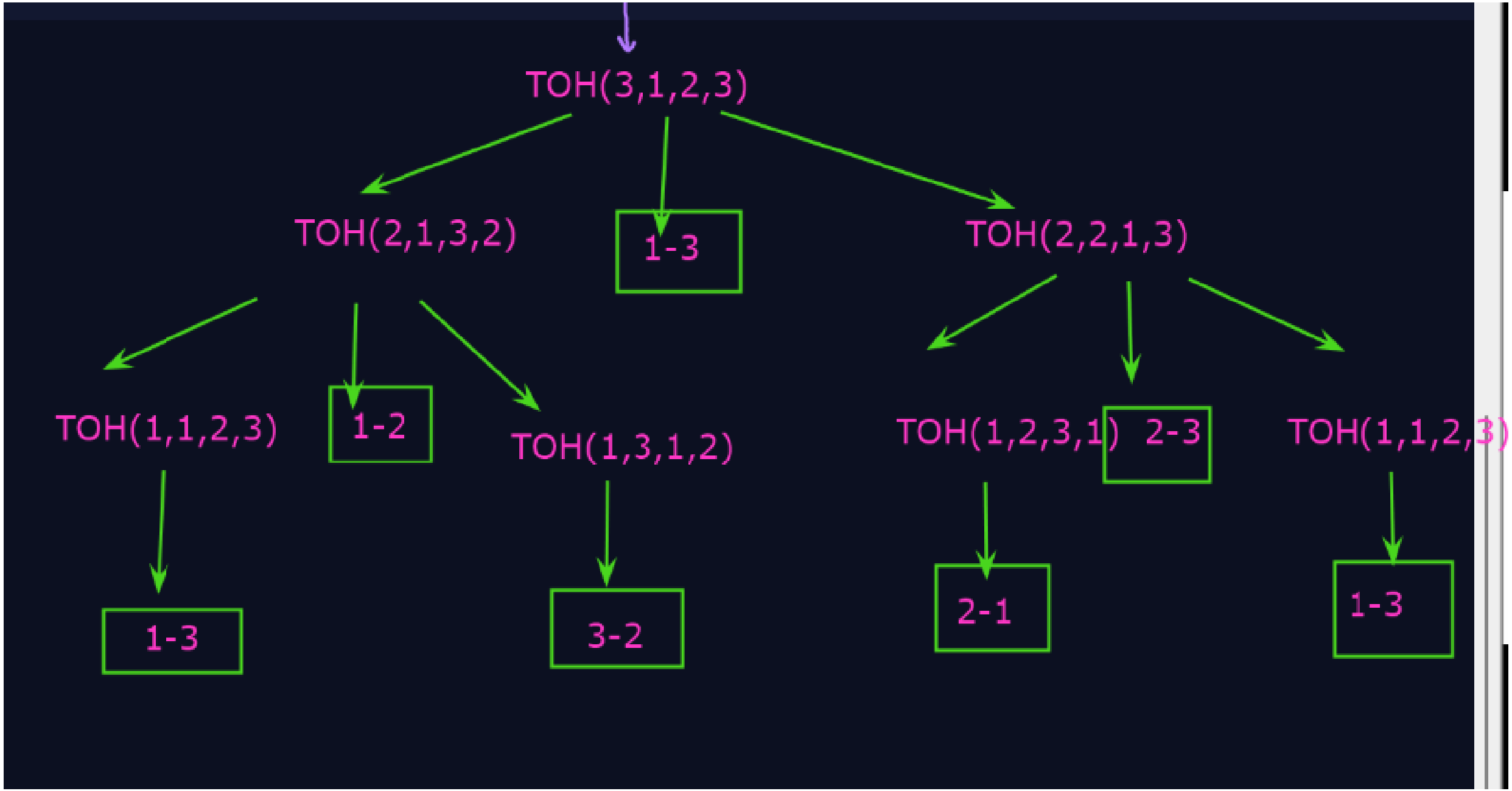
1. Only one disk can be moved among the towers at any given time.
2. Only the "top" disk can be removed.
3. No large disk can sit over a small disk.

Algorithm 1: Recursive algorithm for solving Towers of Hanoi

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



```
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');
    }
}
```


TOH(3,1,2,3)
TOH(2,1,3,2)
1-3
TOH(2,2,1,3)
TOH(1,1,2,3)
1-2
TOH(1,3,1,2)
TOH(1,2,3,1)
2-3
TOH(1,1,2,3)
1-3
3-2
2-1
1-3

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 Recursiona

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.

Problem 1

Recursive program to find the Sum of the series 1 – 1/2 + 1/3 – 1/4 … 1/N
Given a positive integer N, the task is to find the sum of the series 1 – (1/2) + (1/3) – (1/4) +…. (1/N) using recursion.

Examples:

Input: N = 3
Output: 0.8333333333333333
Explanation:
1 – (1/2) + (1/3) = 0.8333333333333333

Input: N = 4
Output: 0.5833333333333333
Explanation:
1- (1/2) + (1/3) – (1/4) = 0.5833333333333333

Problem 2

Recursive Program to print multiplication table of a number
Given a number N, the task is to print its multiplication table using recursion.
Examples

Input: N = 5
Output:
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50

Input: N = 8
Output:
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80

Problem 4

Java Program to Reverse a Sentence Using Recursion

A sentence is a sequence of characters separated by some delimiter. This sequence of characters starts at the 0th index and the last index is at len(string)-1. By reversing the string, we interchange the characters starting at 0th index and place them from the end. The first character becomes the last, the second becomes the second last, and so on.

Example:

Input : CDACMumbai
Output: iabmuMCADC

Input : Alice
Output: ecilA
Approach:

Problem 4: Find duplicates in an array

- Given an array a1[] of size N which contains elements from 0 to N-1, you need to find all the elements occurring more than once in the given array.

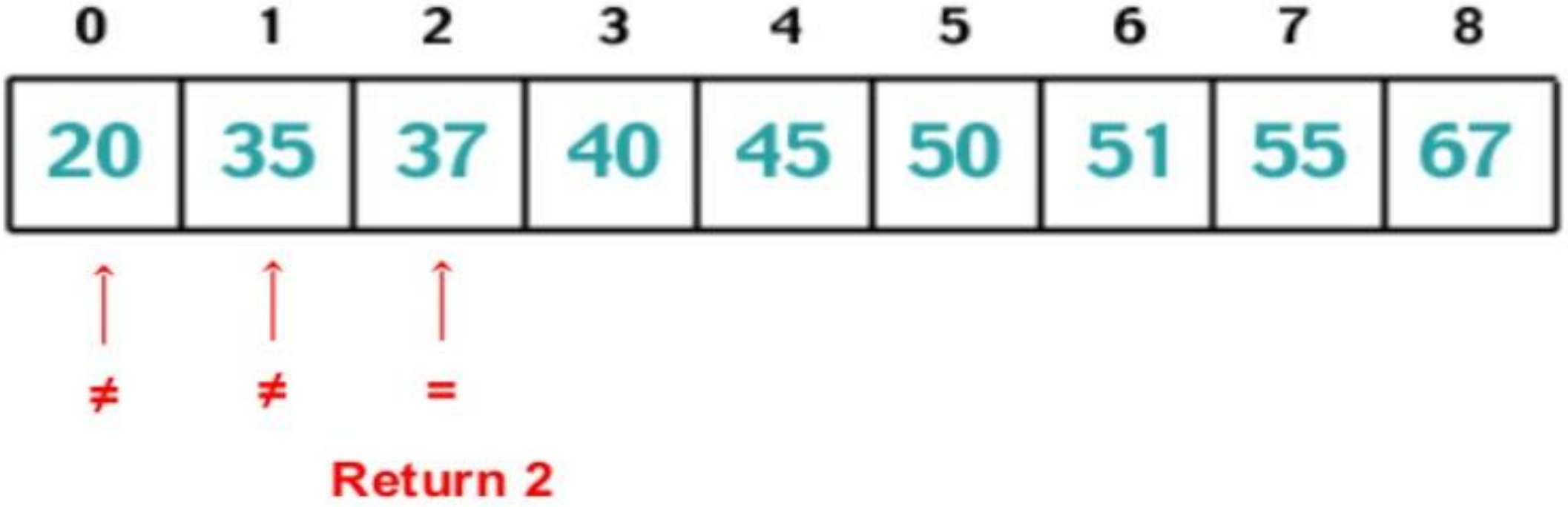
- **Example 1:**
  - Input:
    - N = 4
    - a[] = {0,3,1,2}
  - Output: -1
  - Explanation: N=4 and all elements from 0 to (N-1 = 3) are present in the given array. Therefore output is -1.

- **Example 2:**
  - Input:
    - N = 5
    - a[] = {2,3,1,2,3}
  - Output: 2 3
  - Explanation: 2 and 3 occur more than once in the given array.

Collection of data in arrays

Linear Search

- Find 37?



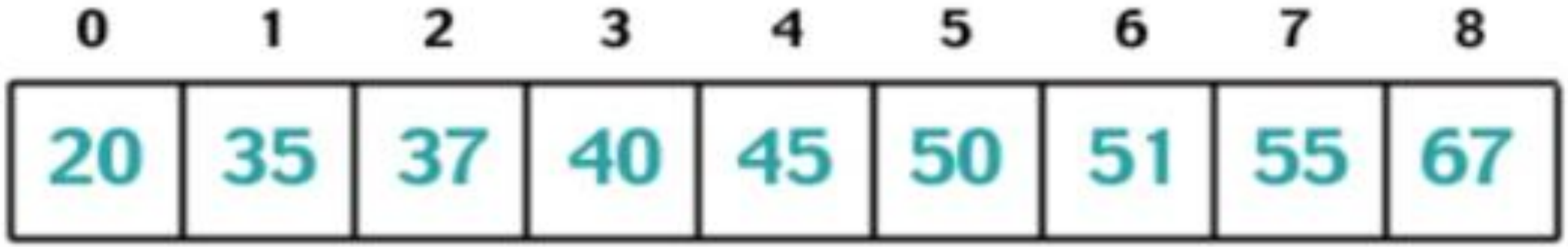
Program 5

Problem: Given an array arr[] of n elements, write a function to search a given element x in arr[].

Examples :

Input : arr[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170}
x = 110;
Output : 6
Element x is present at index 6

Input : arr[] = {10, 20, 80, 30, 60, 50,
110, 100, 130, 170}
x = 175;
Output : -1
Element x is not present in arr[].

Binary Search

- Find 37?
  1. Sort Array.