# Recursion:

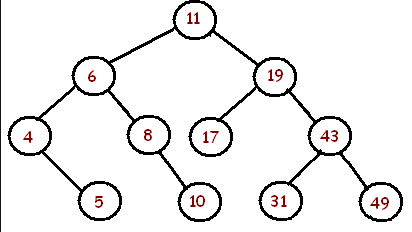
Properties:

* Same operation over and over again with different input.
* Every step the problem gets smaller.
* We need to have a base condition on when to stop the action. Else system will not know when to stop the action.

Similar operation can be done through iteration.

**Example of a gift box.**

## Algorithm using recursion:



Search (root, Value)

If (root equal null)

Return null

Else if (root equal to Value )

Return root.

Else if (value < root)

Search (root.left, value)

Else

Search (root.right, value)

Search 49:

STEP 1: right-> 19

STEP 2: right -> 43

STEP 3: Right-> 49

STEP 4: Found.

# Why Recursion:

* Easy to read and write.
* Always breaking down.
* Difference between loop and recursion.
* Done when we have chance to breakdown.
* Only if a smaller problem is similar to its bigger problem.
* Extendedly used in tree and graphs.
* Also used in several Algorithm.

# Format to use Recursion:

**Format of recursive function:** Recursive case and a base case(Where to stop).

**Base Case:** Case where the function will not follow recursion.

SampleRecursion(Parameter)

If (Base case is satisfied)

Return to base case value.

Else

Return SampleRecursion(Modified parameter)

# How Recursive method works internally:

A():

B()

Print(“A”)

B():

C()

Print(“B”)

C():

D()

Print(“C”)

Stack: Push(Going in) and pop(Going out)

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Examples of Recursive Function:

## Factorial:

5! = 5\*4\*3\*2\*1

Factorial (n)

If (n<=0)

Return null

Else

Return (n\* Factorial (n-1)

## Fibonacci:

0, 1, 1, 2, 3, 5, 8

Function (n)

If (n<=0)

Return null

Else

Logic

Return (Function (n-1)

# Recursion vs Iteration:

Any problem that can be solved by recursion can be solved by Iteration.

|  |  |  |
| --- | --- | --- |
| Particular | Recursion | Iteration |
| Space Efficient | No | Yes |
| Time Efficient | No | Yes |
| Ease of Coding | Yes | No |

# Advantage, Disadvantage and Application.

## When to use Recursion:

When it is possible to break the problem into sub problem.

When it fine if we don’t have constrain about space and time.

When we need to have a solution quick.

## When to avoid:

If a problem cannot be broken down

Or if we have time or space constrain.

## Practical use:

Stack

Tree – Insert/Search/Deletion/Traversal

Sorting

Divide and Conquer

Dynamic Programming