# What is an Algorithm and what makes it good one?

Algorithm is a set of instructions that need to be performed to complete the task.

* Correctness: It should solve the problem correctly
* Efficiency: it should solve the problem efficiently

# WHY DSA IMPORTANT?

Data -> Process -> Output: We must optimize the entire process. We do it using the DSA.

# Types of Data Structure

* Primitive: Integer, Float, Character, Boolean
* Non primitive: Linear or Non-linear
  + Linear: Static(Arrays) or Dynamic(Linked list, Stack, Queue)
  + Non-Linear: Tree, Graph

# Types of Algorithms

* Simple recursive
* Divide and Conquer: Quick sort, Merge sort
* Dynamic Programming: Memoization, remember past result and find new one
* Greedy algorithm: Find the best solution without caring about future. Select global optimum solution
* Brute force algorithm: Find all possibilities
* Randomized algorithm: Uses a random number at least once to decide something.

# Recursion

Function that calls itself. Way of solving problem by having function calling itself.

We try to make the input smaller to make the problem smaller.

***Base condition*** is needed to stop the recursion, otherwise infinite loop will occur.

**Why we need it? :**

* Helps us break down big problem into smaller
* Easy to read than iterative solution

**When to use?**

* When we can divide the problem into smaller subproblem
* To calculate Nth something, code to list “Nth”, “compute all”

Used a lot in **Trees** and **Graphs.**

Used in many algorithms(**divide and conquer, greedy, and Dynamic programming)**

# Logic behind the recursion

1. A method that calls itself
2. Condition to exit from infinite loop

**static** String recursionmethod(String[] params) {

**if**(condition) {

**return** something;

}**else** {

recursionmethod(modified\_params)

}

}

We just must modify everything we have written in the code block above.

System pushed the first method to the “***stack memory”*** to remember its name, then it comes back to the method to finish remaining lines. Stack memory operates on **“Last in First Out (LIFO)”.**

|  |
| --- |
|  |
| thirdMethod() |
| secondMethod() |
| firstMethod() |

Table 1 Stack memory

Sample code below:

Graphical user interface, text, application

Description automatically generated

Java inserts the method with the parameter to the stack.

May cause “***Stack overflow***” exception.

# Recursive vs Iterative Solution

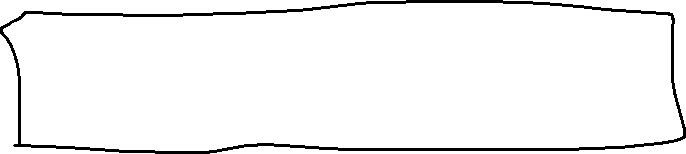
All the recursive solutions can be implemented iteratively. Sometimes it may be too hard to do it.

**Recursion** is very useful for **“*tree traversal*”.**

**Memorization** can reduce the time complexity.

# Write recursion in 3 steps

1. Find the recursive flow



1. Base case: find out the stopping condition
2. Find unintentional case: it should stop for any input

