Recursive Algorithms

Recursion is a technique used in computer science to solve big problems by breaking them into smaller, similar problems. The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. Using a recursive algorithm, certain problems can be solved quite easily.

Introduction to Recursion

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

- A recursive algorithm takes one step toward solution and then recursively call itself to further move. The algorithm stops once we reach the solution.
- Since called function may further call itself, this process might continue forever. So it is essential to provide a base case to terminate this recursion process.

Steps to Implement Recursion

Step1 - Define a base case: Identify the simplest (or base) case for which the solution is known or trivial. This is the stopping condition for the recursion, as it prevents the function from infinitely calling itself.

Step2 - Define a recursive case: Define the problem in terms of smaller subproblems. Break the problem down into smaller versions of itself, and call the function recursively to solve each subproblem.

Step3 - Ensure the recursion terminates: Make sure that the recursive function eventually reaches the base case, and does not enter an infinite loop.

Step4 - Combine the solutions: Combine the solutions of the subproblems to solve the original problem.

Example 1 : Sum of Natural Numbers (n=3)

Input : n = 3

Output : 6

Explanation : The sum of first 3 natural numbers is 1+2+3 = 6.

Input : n = 7

Output : 28

Explanation : The sum of first 7 natural numbers is 1+2+3+4+5+6+7 = 28.

- **Base Case :** At n == 1, it returns 1 for n = 3, the recursion reaches this after going through 3 → 2 → 1.
- **Recursive Case :** Each call adds n to sum(n-1) , so sum(3) = 3 + sum(2), sum(2) = 2 + sum(1).

Example 2 : Factorial of a Number

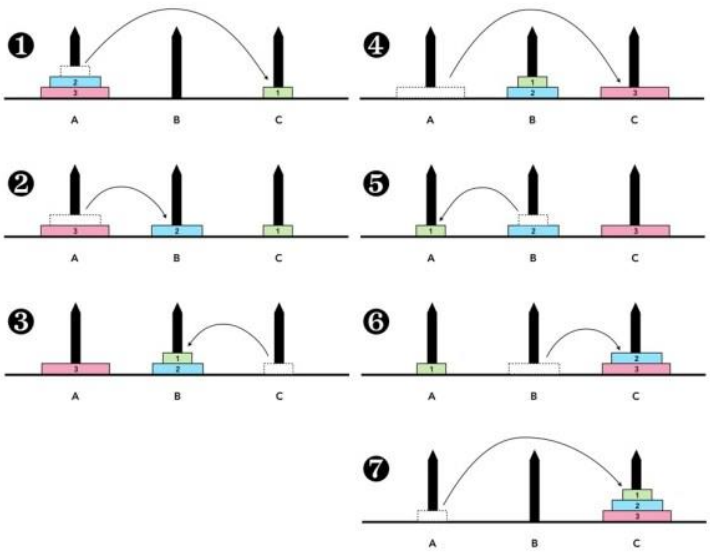
The factorial of a number n (where n >= 0) is the product of all positive integers from 1 to n. To compute the factorial recursively, we calculate the factorial of n by using the factorial of (n-1). The base case for the recursive function is when n = 0, in which case we return 1.

Tower of Hanoi

The Tower of Hanoi is a mathematical puzzle with three poles and stacked disks of different sizes.

The goal is to move all disks from the source pole to the destination pole using an auxiliary pole, following two rules:

1. Only one disk can be moved at a time.
2. A larger disk cannot be placed on a smaller one.



We can then solve the Towers of Hanoi problem for a stack of discs of height n, by trying to solve it for a stack of height $n-1$. To solve for a stack of height $n-1$, we try to solve for a stack of height $n-2$, etc. Now this can be easily solved using a recursive algorithm, because each smaller stack is a sub-problem of the original. To move n discs from tower A to tower C, using tower B as the intermediary, a rudimentary algorithm would look like this:

1. Move $n-1$ discs from A to B.
2. Move one disc from A to C.
3. Move $n-1$ discs from B to C.

Common Applications of Recursion

1. **Tree and Graph Traversal**: Used for systematically exploring nodes/vertices in data structures like trees and graphs.
2. **Sorting Algorithms**: Algorithms like quicksort and merge sort divide data into subarrays, sort them recursively, and merge them.
3. **Divide-and-Conquer Algorithms**: Algorithms like binary search break problems into smaller subproblems using recursion.
4. **Fractal Generation**: Recursion helps generate fractal patterns, such as the Mandelbrot set, by repeatedly applying a recursive formula.
5. **Backtracking Algorithms**: Used for problems requiring a sequence of decisions, where recursion explores all possible paths and backtracks when needed.
6. **Memorization**: Involves caching results of recursive function calls to avoid recomputing expensive subproblems.