What is Recursion?

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. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc. A recursive function solves a particular problem by calling a copy of itself and solving smaller subproblems of the original problems. Many more recursive calls can be generated as and when required. It is essential to know that we should provide a certain case in order to terminate this recursion process. So we can say that every time the function calls itself with a simpler version of the original problem.

Need of Recursion

Recursion is an amazing technique with the help of which we can reduce the length of our code and make it easier to read and write. It has certain advantages over the iteration technique which will be discussed later. A task that can be defined with its similar subtask, recursion is one of the best solutions for it. For example; The Factorial of a number.

Properties of Recursion:

- Performing the same operations multiple times with different inputs.
- In every step, we try smaller inputs to make the problem smaller.
- Base condition is needed to stop the recursion otherwise infinite loop will occur.

What is the difference between direct and indirect recursion?

A function fun is called direct recursive if it calls the same function fun. A function fun is called indirect recursive if it calls another function say fun_new and fun_new calls fun directly or indirectly. The difference between direct and indirect recursion

Recursion	Iteration
Used with functions.	Used with loops.
Every recursive call needs extra space in the stack memory.	Every iteration does not require any extra space.
Smaller code size.	Larger code size.
Terminates when the base case becomes true.	Terminates when the condition becomes false.

Example: Real Applications of Recursion in real problems

Recursion is a powerful technique that has many applications in computer science and programming. Here are some of the common applications of recursion:

- **Tree and graph traversal**: Recursion is frequently used for traversing and searching data structures such as trees and graphs. Recursive algorithms can be used to explore all the nodes or vertices of a tree or graph in a systematic way.
- **Sorting algorithms**: Recursive algorithms are also used in sorting algorithms such as quicksort and merge sort. These algorithms use recursion to divide the data into smaller subarrays or sublists, sort them, and then merge them back together.
- **Divide-and-conquer algorithms**: Many algorithms that use a divide-and-conquer approach, such as the binary search algorithm, use recursion to break down the problem into smaller subproblems.
- **Fractal generation**: Fractal shapes and patterns can be generated using recursive algorithms. For example, the Mandelbrot set is generated by repeatedly applying a recursive formula to complex numbers.
- **Backtracking algorithms**: Backtracking algorithms are used to solve problems that involve making a sequence of decisions, where each decision depends on the previous ones. These algorithms can be implemented using recursion to explore all possible paths and backtrack when a solution is not found.

10. What are the limitations of Recursion?

Limitations of recursion in data structure are:

1. Recursive functions are usually slower compared to non-recursive functions.
2. It might need a considerable amount of memory space to hold intermediate outcomes on the system stacks.
3. Difficult to assess or grasp the code.
4. It's not very effective in terms of space and time complexity.
5. The laptop or computer might run out of memory in case the recursive calls aren't correctly checked.

5. What are the basic rules of Recursion?

Just like the robots of Asimov, all recursive algorithms have to obey 3 significant laws:

1. A recursive algorithm has to call itself, recursively.
2. A recursive algorithm requires a base case.
3. A recursive algorithm must alter its move and state toward the base case.

ABDULLAH

Fibonacci Sequence

The Fibonacci Sequence is the series of numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

The next number is found by adding up the two numbers before it:

- the 2 is found by adding the two numbers before it (1+1),
- the 3 is found by adding the two numbers before it (1+2),
- the 5 is (2+3),
- and so on!

Example: the next number in the sequence above is 21+34 = **55**

The Fibonacci sequence is **a set of integers (the Fibonacci numbers) that starts with a zero, followed by a one, then by another one, and then by a series of steadily increasing numbers**. The sequence follows the rule that each number is equal to the sum of the preceding two numbers.

Summary of Recursion:

- There are two types of cases in recursion i.e. recursive case and a base case.
- The base case is used to terminate the recursive function when the case turns out to be true.
- Each recursive call makes a new copy of that method in the stack memory.
- Infinite recursion may lead to running out of stack memory.
- Examples of Recursive algorithms: Merge Sort, Quick Sort, Tower of Hanoi, Fibonacci Series, Factorial Problem, etc.

How are recursive functions stored in memory?

Recursion uses more memory, because the recursive function adds to the stack with each recursive call, and keeps the values there until the call is finished. The recursive function uses LIFO (LAST IN FIRST OUT) Structure just like the stack data structure.