Dated:

RECURSION:

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive algorithm, certain problems can be solved quite easily. such as i.e postorder / preorder / inorder tranversals and many more.

Importance:

Recursion is an amazing technique with the help of which we can reduce length of our code and make it easier to read and write. It has certain advantages over iteration such as;

Recursion	Iteration
• Terminates when the base case becomes true.	• Terminates when the condition becomes false.
• 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.

Base Condition:

In the recursive program, the solution to the base case is provided and the solution to the bigger problem is expressed in terms of smaller problems.

Dated:

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.

Storing 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.

Direct Recursion:

A function fn is called direct recursive function if it calls the same function fn.

Indirect Recursion:

A function fn is called indirect recursive function if it calls the another function say fn_new.

Tailed / Non Tailed Recursion:

A recursive function is tail recursive when a recursive call is the last thing executed by the function.

Problem Solving Using Recursion:

The idea is to represent a problem in terms of one or more smaller problems, and add one or more smaller problems, and add one or more base conditions that stop the recursion.

Dated:

PROBLEM: Write a program and recurrence relation to find the Factorial of n where n > 2.

Mathematical approach:

if n == 0 or n == 1:
f(n) = n * f(n-1) if n > 1:

Recurrence Relation:

T(n) = 1 for n = 0
T(n) = 1 + T(n-1) for n > 0

Code:

```
1  def Fact(n):
2      if n==0 or n == 1:
3          return 1:
4      else:
5          return n * fact(n-1):
6
7  n = int(input("Enter a number:"))
8  f = Fact(n)
9  print("Factorial of", n, "is:", f")
```

Explanation:

As we can see that line 7, 8, 9 are the lines of main rest are the lines of code are of function Fact(n). Line 5 we getting an recursive function Fact(n-1) recalling from Fact(n)

Output:

```
Enter a number: 5
Factorial of 5 is: 120
```

Dated:

Disadvantages of Recursion over iterative:

The recursive program has greater space requirements than the iterative as all functions will remain in the stack until the base case is reached. It also has greater time requirement because of function calls and return overhead.
Moreover, due to the smaller length of code, the codes are difficult to understand and hence extra care has to be practised while writing the code. The computer may run out of memory if the recursive calls are not properly checked.

Advantages of Recursion over Iteration:

Recursion provides a clean and simple way to write a code. Some problems are inherently recursive like tree transversal etc. For such problems, it is prefered to write recursive code. we can write such codes also iteratively with the help of a stack data structure.

Important points to be noted:

- There are two types of cases in recursion i.e base and recursive.
- 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.
- Infinite recursion may lead to running out of stack memory.
- Examples of Recursive algorithm:
  Merge sort, Quick sort, Tower of Hanoi, Fibanacci series, Factorial etc.

Recursive Control

The binary search algorithm is similar to the sequential search in that each algorithm requests the execution of a repetitive process. However, the implementation of this repetition is significantly different. Whereas the sequential search involves a circular form of repetition, the binary search executes each stage of the repetition as a subtask of the previous stage. This technique is known as **recursion.**

As we have seen, the illusion created by the execution of a recursive procedure is the existence of multiple copies of the procedure, each of which is called an activation of the procedure. These activations are created dynamically in a telescoping manner and ultimately disappear as the algorithm advances. Of those activations existing at any given time, only one is actively progressing. The others are effectively in limbo, each waiting for another activation to terminate before it can continue.

Being a repetitive process, recursive systems are just as dependent on proper control as are loop structures. Just as in loop control, recursive systems are dependent on testing for a termination condition and on a design that ensures this condition will be reached. In fact, proper recursive control involves the same three ingredients—initialization, modification, and test for termination—that are required in loop control.

In general, a recursive procedure is designed to test for the termination condition (often called the **base case** or **degenerative case**) before requesting further activations. If the termination condition is not met, the routine creates another activation of the procedure and assigns it the task of solving a revised problem that is closer to the termination condition than that assigned to the current activation. However, if the termination condition is met, a path is taken that causes the current activation to terminate without creating additional activations.

Let us see how the initialization and modification phases of repetitive control are implemented in our binary search procedure of Figure 5.14. In this case, the creation of additional activations is terminated once the target value is found or the task is reduced to that of searching an empty list. The process is initialized implicitly by being given an initial list and a target value. From this initial configuration the procedure modifies its assigned task to that of searching a smaller list. Since the original list is of finite length and each modification step reduces the length of the list in question, we are assured that the target value ultimately is found or the task is reduced to that of searching the empty list. We can therefore conclude that the repetitive process is guaranteed to cease.

Finally, since both loop and recursive control structures are ways to cause the repetition of a set of instructions, we might ask whether they are equivalent in power. That is, if an algorithm were designed using a loop structure, could another algorithm using only recursive techniques be designed that would solve the same problem and vice versa? Such questions are important in computer science because their answers ultimately tell us what features should be provided in a programming language in order to obtain the most powerful programming system possible. We will return to these ideas in Chapter 12 where we consider some of the more theoretical aspects of computer science and its mathematical foundations. With this background, we will then be able to prove the equivalence of iterative and recursive structures in Appendix E.