Recursion – Factorial Using Recursion

1. Introduction

The **factorial of a number** is a classic problem used to understand and demonstrate **recursion**.

It is one of the best examples because the problem naturally **breaks itself into smaller sub-problems**, which is the core idea of recursion.

---

2. What is Factorial?

The factorial of a non-negative integer n is the product of all positive integers less than or equal to n.

Mathematical Definition:

```
n! = n × (n−1) × (n−2) × ... × 1
```

Examples:

- 0! = 1
- 1! = 1
- 5! = 5 × 4 × 3 × 2 × 1 = 120

---

3. Why Factorial is Suitable for Recursion

Factorial is ideal for recursion because:

- The problem is defined in terms of itself
- n! depends on (n−1)!
- The input size reduces with each call

- A clear base case exists

---

4. Recursive Definition of Factorial

Factorial can be defined recursively as:

```
n! = n × (n−1)!
```

This definition clearly shows:

- **Recursive case:** n × factorial(n−1)
- **Base case:** when n becomes 0 or 1

---

5. Base Case for Factorial

The **base case** stops the recursion.

Base Case Condition:

- If n == 0 or n == 1, return 1

This is because:

- 0! = 1
- 1! = 1

---

6. Recursive Case for Factorial

The **recursive case** reduces the problem size.

Recursive Step:

- Multiply the current number n with the factorial of (n−1)

Each recursive call moves closer to the base case.

---

7. Logic for Factorial Using Recursion (Plain English)

1. If the number is 0 or 1, return 1

2. Otherwise, multiply the number with factorial of (number − 1)
3. Repeat until the base case is reached
4. Return results back through the call stack

---

8. Step-by-Step Execution Example

Find factorial of 4:

```
factorial(4)
= 4 × factorial(3)
= 4 × 3 × factorial(2)
= 4 × 3 × 2 × factorial(1)
= 4 × 3 × 2 × 1
= 24
```

---

9. Call Stack Visualization

```
factorial(4)
factorial(3)
factorial(2)
factorial(1) → returns 1
```

Then values return back:

```
2 × 1 → 2
3 × 2 → 6
4 × 6 → 24
```

---

10. Time and Space Complexity

Aspect	Complexity
Time Complexity	O(n)

Aspect	Complexity
Space Complexity	O(n)

Space is used due to recursive function calls stored in the call stack.

---

11. Advantages of Recursive Factorial

- Simple and elegant logic
- Easy to understand recursion flow
- Matches mathematical definition closely
- Useful for learning recursion concepts

---

12. Limitations of Recursive Factorial

- Uses extra memory due to call stack
- Slower compared to iterative solution
- Risk of stack overflow for large inputs

---

13. Iterative vs Recursive Factorial

Aspect	Recursive	Iterative
Code Readability	High	Medium
Performance	Slower	Faster
Memory Usage	High	Low
Learning Purpose	Excellent	Good

---

14. Real-World Relevance

- Used in mathematics and combinatorics
- Helps understand recursion flow
- Basis for permutations and combinations

- Common exam and interview question

---

15. Summary

- Factorial is a classic recursion example
- Uses base case and recursive case
- Each call reduces input size
- Time complexity is O(n)
- Important for understanding recursion
