Recursion Basics – Summary

1. Introduction

Recursion is a programming technique where a function **calls itself** to solve a problem. It is often used to **break complex problems into smaller, simpler subproblems**.

Key points: - Every recursive function must have a **base case** to stop recursion. - Recursive calls should progress towards the base case. - Helps in solving problems like factorial, Fibonacci, tree traversals, and more.

2. Components of Recursion

1. **Base Case**: The condition under which the recursion stops.
2. **Recursive Case**: The part where the function calls itself with modified parameters.

Example (Factorial):

```
int factorial(int n) {
    if(n == 0) return 1; // Base case
    return n * factorial(n-1); // Recursive case
}
```

Example (Fibonacci):

```
int fibonacci(int n) {
    if(n <= 1) return n; // Base case
    return fibonacci(n-1) + fibonacci(n-2); // Recursive case
}
```

3. Types of Recursion

- **Direct Recursion**: Function calls itself directly.
- **Indirect Recursion**: Function A calls Function B, which calls Function A.
- **Tail Recursion**: Recursive call is the last operation in the function.

```
int sum(int n, int acc) {
    if(n == 0) return acc;
    return sum(n-1, acc+n); // Tail recursion
}
```

- **Non-Tail Recursion**: Recursive call is not the last operation.

```
int factorial(int n) {
    if(n == 0) return 1;
    return n * factorial(n-1); // Non-tail recursion
}
```

4. Advantages of Recursion

- Simplifies code for problems with repetitive substructure.
- Makes code more readable and easier to understand for divide-and-conquer problems.

5. Disadvantages of Recursion

- Can be **less efficient** than iteration due to function call overhead.
- May cause **stack overflow** if recursion is too deep.

6. Common Examples

- Factorial calculation (see above)
- Fibonacci series (see above)
- Tree traversals (preorder, inorder, postorder)
- Graph traversal (DFS)
- Towers of Hanoi

7. Tips for Writing Recursion

1. Always define a **base case**.
2. Ensure **recursive calls move towards base case**.
3. Test with small input first.
4. Consider **iteration** if recursion depth is too large.

8. Summary Table

Component	Description	Example
Base Case	Stops recursion	factorial(0) = 1
Recursive Case	Calls itself with smaller problem	factorial(n) = n*factorial(n-1)
Tail Recursion	Last action is recursive call	sum(n, acc)
Non-Tail Recursion	Recursive call not last operation	factorial(n)

Key Points: - Recursion breaks problems into smaller subproblems. - Always have a base case to avoid infinite recursion. - Useful for problems with repetitive or hierarchical structure. - Can be replaced by iteration for efficiency if needed.