**Recursion in Java**

**Definition**:

Recursion is a technique where a method calls itself to solve a problem by dividing it into smaller instances of the same problem. It requires a base case to terminate the recursion.

**Basic Structure:**

1. Base Case: Stops recursion and returns a result.

2. Recursive Case: Method calls itself with modified parameters.

Example: Factorial Calculation

public class RecursionExample {

public static int factorial(int n) {

if (n == 0 || n == 1) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

public static void main(String[] args) {

int number = 5;

System.out.println("Factorial of " + number + " is " + factorial(number));

}

}

**Benefits:**

- Divide and Conquer: Simplifies complex problems by breaking them into smaller parts (e.g., sorting algorithms).

- Code Simplicity: Makes code more readable for problems like tree traversals.

**Considerations:**

- Base Case: Must be correctly defined to avoid infinite recursion.

- Performance: Can be less efficient and use more stack space than iterative methods.

**Time Complexity of Recursive Algorithm**

**Example: Factorial Calculation**

public static int factorial(int n) {

if (n == 0 || n == 1) return 1;

else return n \* factorial(n - 1);

}

**Time Complexity:**

- Factorial Calculation: O(n), since each call results in one more recursive call.

**Optimizing Recursive Solutions**

1. **Memorization**:

- Description: Store results of previous computations to avoid redundant calculations.

- Example:

import java.util.HashMap;

public static int factorial(int n) {

if (n == 0 || n == 1) return 1;

if (memo.containsKey(n)) return memo.get(n);

int result = n \* factorial(n - 1);

memo.put(n, result);

return result;

}

2. **Tail Recursion**:

- Description: Refactor recursion so the recursive call is the last action. Some optimizers can handle this more efficiently.

- Example:

private static int factorialHelper(int n, int acc) {

if (n == 0) return acc;

return factorialHelper(n - 1, n \* acc);

}

3. **Iterative Approach**:

- Description: Use an iterative loop to avoid recursion overhead.

- Example:

public static int factorial(int n) {

int result = 1;

for (int i = 1; i <= n; i++) result \*= i;

return result;

}

Summary:

Recursion for factorial has O(n) time complexity. Optimize by using memoization to store results, tail recursion to reduce stack depth, or an iterative approach to eliminate recursion overhead.