Mastering Recursion: Beginner to Advanced

1. Factorial of a Number (Basic Recursion)

Problem: Find the factorial of a given number n.

Code:

```
function factorial(n) {
    if (n === 0) return 1;
    return n * factorial(n - 1);
}
```

Program Flow:

- Input n=4.
- factorial(4) = 4 * factorial(3)
    factorial(3) = 3 * factorial(2)
        factorial(2) = 2 * factorial(1)
            factorial(1) = 1 * factorial(0)
                factorial(0) = 1 (Base case)
            = 1 * 1 = 1
        = 2 * 1 = 2
    = 3 * 2 = 6
= 4 * 6 = 24

Final Answer = 24

2. Fibonacci Series (Basic Recursion with Multiple Calls)

Problem: Find nth Fibonacci number.

Code:

```
function fibonacci(n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
```

```
}
```

Program Flow Example (n=5):

fibonacci(5) = fibonacci(4) + fibonacci(3)

fibonacci(4) = fibonacci(3) + fibonacci(2)

fibonacci(3) = fibonacci(2) + fibonacci(1)

... continues until base case n <= 1.

Note: Time Complexity is exponential, O(2^n), use memoization for optimization.

3. Power Function (Optimized Divide and Conquer)

Problem: Compute a^b efficiently.

Code:

```
function power(a, b) {
    if (b === 0) return 1;
    let half = power(a, Math.floor(b/2));
    if (b % 2 === 0) return half * half;
    return a * half * half;
}
```

Program Flow (Example a=2, b=5):

power(2,5) -> power(2,2) -> power(2,1) -> power(2,0)=1

Backtrack: power(2,1)=2*1=2, power(2,2)=2*2=4, power(2,5)=2*4*4=32

4. Subset Generation (Backtracking Recursion)

Problem: Print all subsets of a set.

Code:

```
function subset(arr, index, current) {
    if (index === arr.length) {
        console.log(current);
        return;
    }
    subset(arr, index+1, current);
    subset(arr, index+1, current.concat(arr[index]));
}
```

Flow Example for [1,2]:

```
subset([1,2],0,[]) -> subset([1,2],1,[]) -> subset([1,2],2,[]) -> print []
                                          -> subset([1,2],2,[2]) -> print [2]
                              -> subset([1,2],1,[1]) -> ...
```

Prints: [], [2], [1], [1,2]

5. Tower of Hanoi (Classic Recursion Problem)

Problem: Move n disks from rod A to rod C using rod B.

Code:

```
function hanoi(n, from, to, aux) {
    if (n === 1) {
        console.log(`Move disk 1 from ${from} to ${to}`);
        return;
    }
    hanoi(n-1, from, aux, to);
    console.log(`Move disk ${n} from ${from} to ${to}`);
    hanoi(n-1, aux, to, from);
}
```

Flow Example for n=3:

hanoi(3, 'A', 'C', 'B') calls:

- Move top 2 disks to B,
- Move disk 3 to C,
- Move 2 disks from B to C.

Total moves = 7.

6. Advanced: Recursive Backtracking - N-Queens Problem

Problem: Count all possible ways to place N queens on N×N board.

Code Idea:

- Place queen row by row.
- For each column, check if it's safe (not attacked by previous queens).
- If valid, move to next row (recursive call).
- Backtrack after each row.

Concept:

- Highly used in competitive programming, interviews.
- Emphasizes recursive tree traversal, backtracking pruning.