Recursion in DSA: Beginner to Expert Guide

Subtitle: Master recursion step by step and learn how to solve problems that seem impossible without it.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Recursion is a method where a function calls itself to solve a problem. Many coding problems in DSA, especially in trees, backtracking, and dynamic programming, require recursion. Beginners often struggle, but once mastered, it becomes a powerful tool.

Step 1: What is Recursion?

Recursion is solving a problem by **reducing it to smaller sub-problems**.

- Every recursive function must have:
  - **Base case** → stops recursion
  - **Recursive case** → calls itself

Example: Factorial

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

Step 2: How to Think Recursively

1. Identify the **smallest sub-problem**
2. Solve it directly (**base case**)
3. Assume recursion solves smaller cases
4. Combine results to solve the original problem

Example: Sum of array

```
function sumArray(arr, n) {
 if (n === 0) return 0;
 return arr[n - 1] + sumArray(arr, n - 1);
```

```
}
```

Step 3: Common Recursive Patterns

1. Simple Recursion

- Factorial, Fibonacci, sum of array, power of a number

2. Backtracking

- Generate all subsets
- N-Queens problem

3. Divide & Conquer

- Merge Sort, Quick Sort
- Binary Search

4. Tree Recursion

- Traverse nodes
- Calculate height, sum, or count nodes

Step 4: Important Problems You Must Practice

1. Factorial

Covered in Step 1

2. Fibonacci Numbers

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

3. Reverse a String

```
function reverseString(str) {
  if (str.length === 0) return "";
  return reverseString(str.slice(1)) + str[0];
}
```

4. Sum of Digits

```
function sumDigits(n) {
  if (n === 0) return 0;
  return (n % 10) + sumDigits(Math.floor(n/10));
}
```

5. Generate Subsets (Backtracking)

```
function subsets(arr, index=0, current=[], result=[]) {
  if (index === arr.length) {
    result.push([...current]);
    return;
  }
  current.push(arr[index]);
  subsets(arr, index+1, current, result); // include
  current.pop();
  subsets(arr, index+1, current, result); // exclude
  return result;
}
```

Step 5: Common Mistakes

- Forgetting **base case** → leads to infinite recursion
- Thinking **iteratively** instead of recursively
- Not understanding **call stack**
- Inefficient recursion (ex: naive Fibonacci)

Step 6: Practice Plan for Recursion

Day 1:

- Factorial, Fibonacci, sum → 5 problems

Day 2:

- Reverse string, sum of digits → 5–7 problems

Day 3–4:

- Backtracking: subsets, combinations → 8–10 problems

Day 5:

- Divide & conquer: merge sort, quick sort → 5 problems

Day 6:

- Mixed recursion + problem solving → 10 problems

Mini Exercises

1. Power of a number using recursion
2. Count occurrences of a digit in a number
3. Generate all permutations of a string
4. Solve Tower of Hanoi problem

Key Takeaways

- Recursion is essential for **trees, backtracking, and divide & conquer problems**
- Always define **base case** first
- Break problems into **smaller sub-problems**
- Practice a variety of problems to **build intuition**
- Understand **call stack behavior** to debug

Visit haas.dev for more DSA recursion guides, problem sets, and interview prep material.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app