DSA Basic - [01] Introduction

Table of Contents

1. Mentor Information
2. What is Data Structures and Algorithms?
3. Time and Space Complexity (Big O Notation)
4. Basic Problem-Solving Techniques
5. How the Judger Works (Auto & Manual)
6. Coding Interview Format
7. Common Mistakes and Debugging Tips

Mentor Information

Quoc Bui

- **Role**: Course Mentor
- **Experience**: Over 5 yoe in DSA.
- **Achievements**:
  - OLP'22 | *Second Prize*
  - IEEEXtreme 17.0 | *Ranked 21/4222*
  - The 2022 ICPC Vietnam National Programming Contest | *Honorable Mention*

Nam Le

- **Role**: Teaching Assistant
- **Achievements**:
  - IEEEXtreme 16.0 | *Ranked 80*
  - The 2022 National Competition for Excellent Students | *Third Prize*
  - The 28th National Youth Informatics Contest | *Third Prize*

How about you?

What is Data Structures and Algorithms?

Problems:

- How do we store and organize data?
- How do we solve problems efficiently?
- How do we analyze the performance of our solutions?

How do we store and organize data?

In programming, we use **data structures** to store and organize data.

Common data structures:

- Array
- Linked List
- Stack
- Queue
- Tree
- Map
- Graph
- ...

How do we solve problems efficiently?

We use **algorithms** to solve problems efficiently.

An algorithm is a step-by-step procedure to solve a problem.

Example:

- **Problem**: Find the maximum number in an array.
- **Algorithm**: Iterate through the array and keep track of the maximum number.

Common algorithms:

- Mathematical algorithms (e.g., Prime numbers, Fibonacci sequence)
- Sorting algorithms (e.g., Bubble Sort, Quick Sort)
- Searching algorithms (e.g., Linear Search, Binary Search)
- Graph algorithms (e.g., Breadth-First Search, Depth-First Search)
- ...

How do we analyze the performance of our solutions?

- **Time complexity**: How long does the algorithm take to run?
- **Space complexity**: How much memory does the algorithm use?

Explore more?

Go to next slide.

Time and Space Complexity (Big O Notation)

Time Complexity

Time complexity is defined as the amount of time taken by an algorithm to run, as a function of the length of the input.

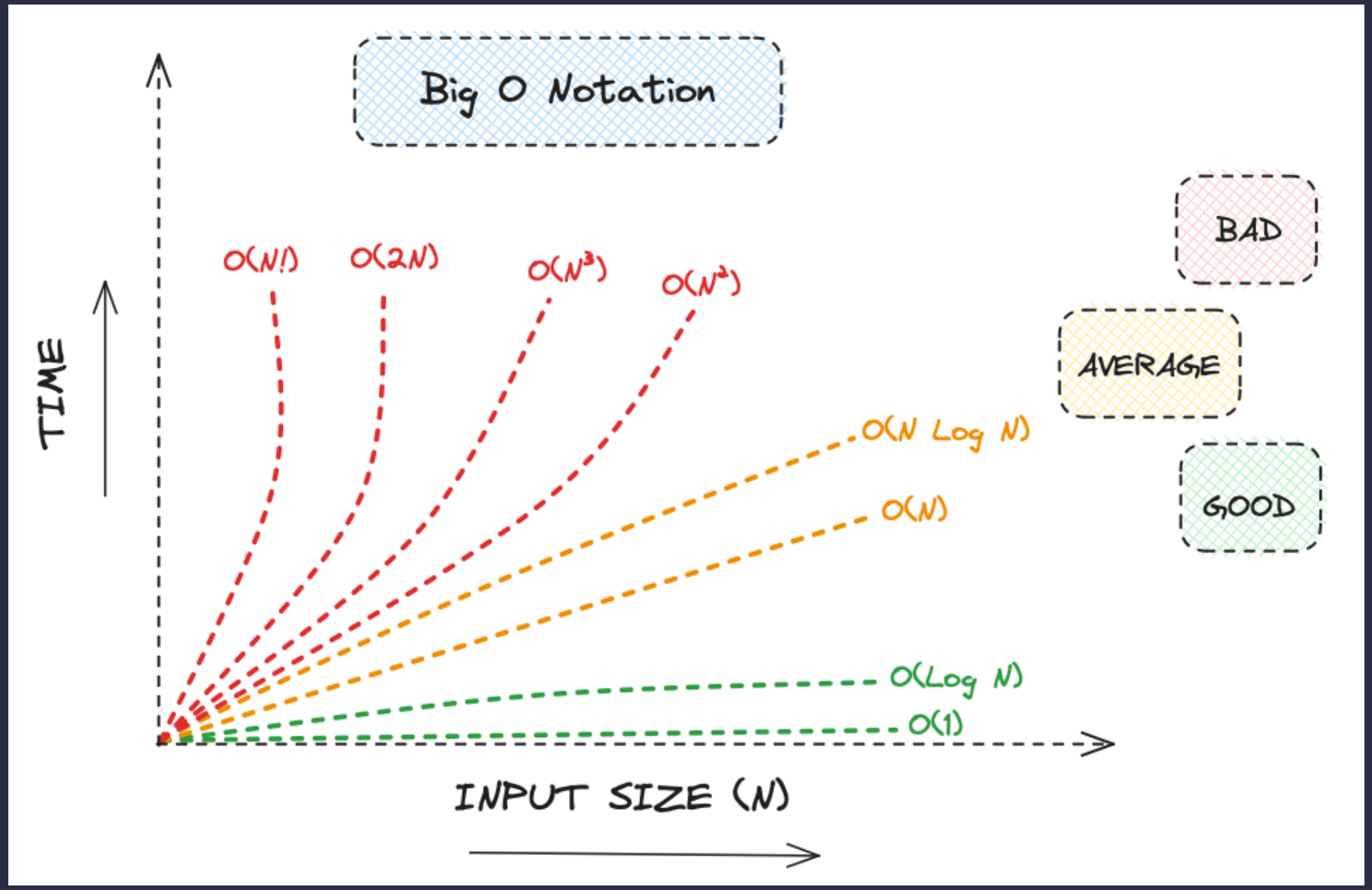
Example:

```
function sum(arr) {
  let total = 0;
  for (let num of arr) {
    total += num;
  }
  return total;
}
```

Time complexity: O(n) (linear time)

Common Time Complexities

- O(1) - Constant time
  - Example: Sum of two numbers
- O(log n) - Logarithmic time
  - Example: Binary search
- O(n) - Linear time
  - Example: Sum of an array
- O(n log n) - Log-linear time
  - Example: Quick sort
- O(n^2) - Quadratic time
  - Example: Loop inside a loop
- O(2^n) - Exponential time
  - Example: Recursive Fibonacci
- O(n!) - Factorial time
  - Example: Permutations


Big O Notation
BAD
AVERAGE
GOOD
O(N!)
O(2N)
O(N³)
O(N²)
O(N Log N)
O(N)
O(Log N)
O(1)
TIME
INPUT SIZE (N)

Space Complexity

Space complexity refers to the total amount of memory space used by an algorithm/program, including the space of input values for execution.

Example

Example 1:

```
function linearSearch(arr, target) {
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] === target) {
      return i; // Return the index of the target element if found
    }
  }
  return -1; // Return -1 if the target is not found in the array
}
```

Time Complexity: O(n)

Space Complexity: O(1)

Example 2:

```
function decimalToBinary(dec) {
  if (dec === 0) return "0"; // Edge case for zero

  let binary = "";
  while (dec > 0) {
    let remainder = dec % 2;
    binary = remainder + binary; // Prepend the remainder to form the binary number
    dec = Math.floor(dec / 2); // Continue with the integer division
  }

  return binary;
}
```

Time Complexity: O(log n)

Basic Problem-Solving Techniques

1. Brute Force

- Solve problems by trying all possible solutions

2. Divide and Conquer

- Break the problem into smaller sub-problems and solve them independently

3. Greedy

- Make the locally optimal choice at each stage

How the Judger Works (Auto & Manual)

- **Automatic Judging**:
  - Code runs in a controlled environment
  - Input/output matching based on predefined test cases
  - Time and memory limits
- **Manual Judging**:
  - Human verification for subjective or complex problems

Coding Interview Format

- **Problem Statement**:
  - Description of the problem
  - Input/output format
  - Limits on input size, time, and memory
  - Example test cases
- **Solution**:
  - Code implementation > Explanation of the approach
  - Time and space complexity analysis
- **Discussion**:
  - Edge cases
  - Optimizations
  - Follow-up questions
  - Bugs and debugging

Common Mistakes and Debugging Tips

Common Mistakes:

- Misunderstanding the problem
- Not considering edge cases
- Overcomplicating the solution
- Ignoring time and space complexity
- ...

Debugging Tips:

- Use print statements to trace execution
- Test with sample inputs
- Analyze the code step by step
- Use a debugger tool
- ...

Thank you!