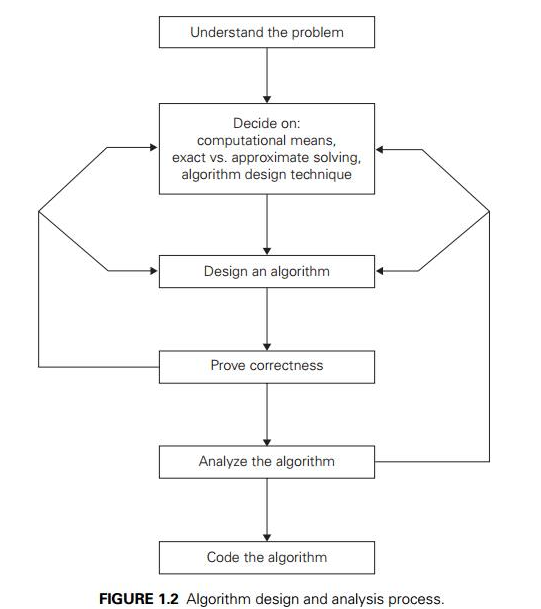
****

**Designing and Analyzing an algorithm.**

**🔷 PART 1: Designing an Algorithm**

This is the creative phase where you figure out how to solve the problem. Here's a structured approach:

**1. Understand the Problem**

1. What are the inputs and expected outputs?
2. Are there constraints (e.g., time, memory)?
3. What are edge cases?

**2. Choose a Strategy**

Common strategies include:

1. **Brute force** – Try all possibilities.
2. **Divide and conquer** – Split into sub-problems (e.g., Merge Sort).
3. **Greedy algorithms** – Make the best choice at each step.
4. **Dynamic programming** – Break into overlapping sub-problems.
5. **Backtracking / recursion** – Explore all possibilities recursively.
6. **Graph algorithms** – BFS, DFS, Dijkstra’s, etc.

**3. Write Pseudocode**

Helps to visualize logic before writing actual code.

**Example:**

**function findMax(arr):**

**max = arr[0]**

**for i from 1 to length(arr)-1:**

**if arr[i] > max:**

**max = arr[i]**

**return max**

**4. Convert Pseudocode to Code**

Use your preferred programming language.

Keep it clean and modular.

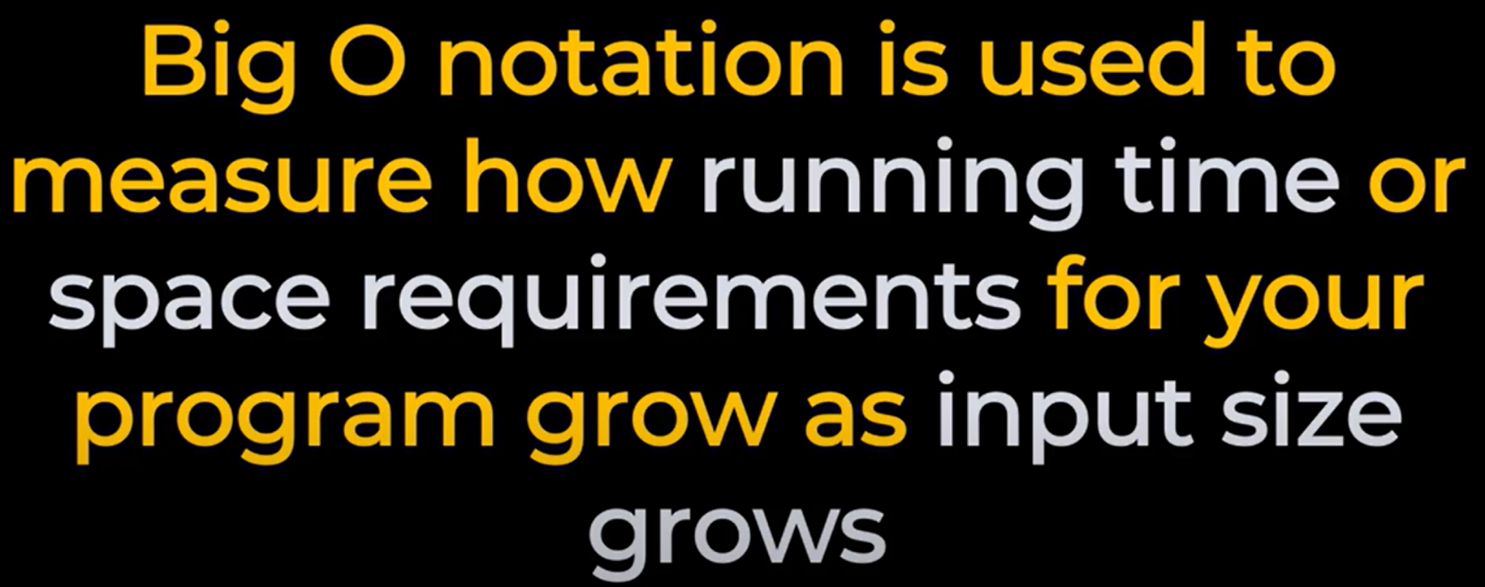
**🔷 PART 2: Analyzing an Algorithm**

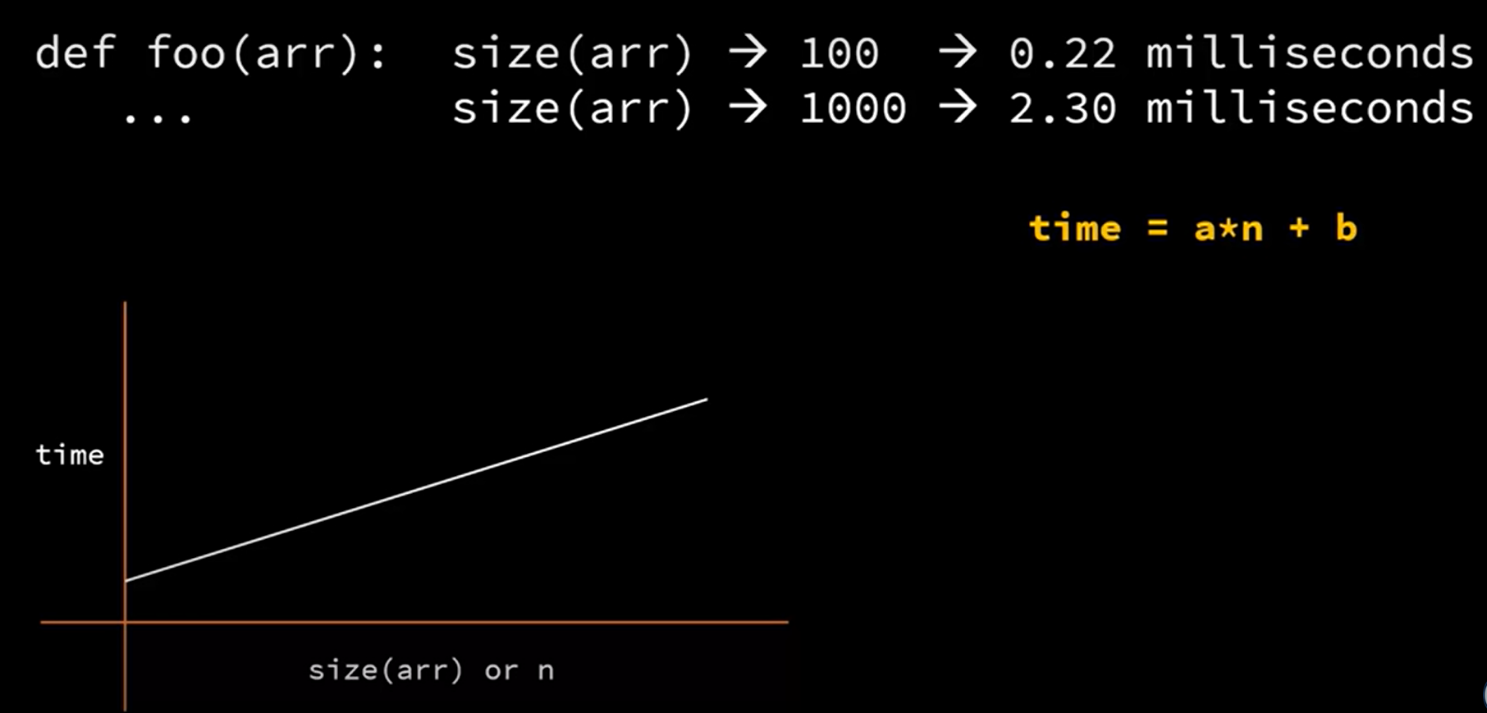
This is the mathematical/logical phase, focusing on performance and correctness.

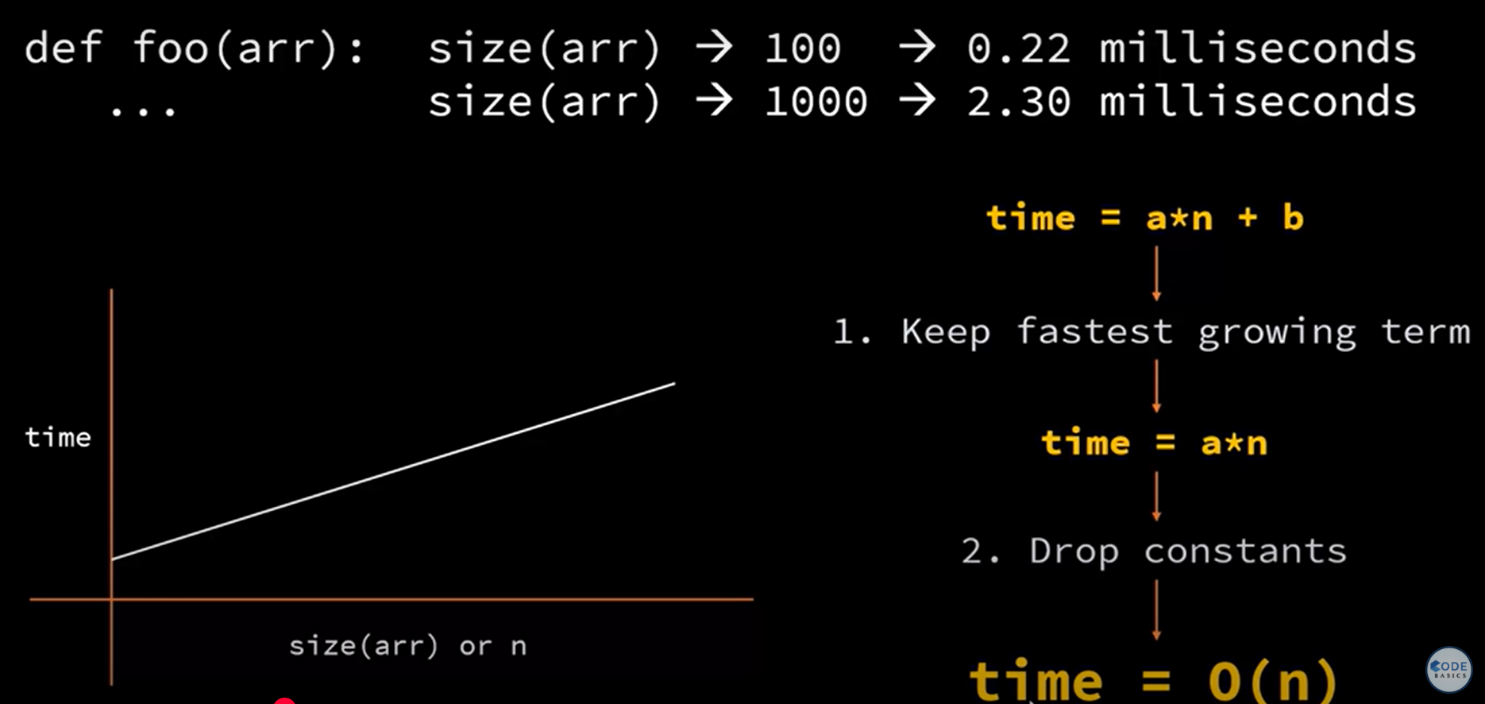
**1. Correctness**

Proof by Induction or Loop Invariants.

Check all test cases, including edge cases.

**2. Time Complexity (Big O Notation) **

****

****

Measure how time grows with input size n.

Common complexities:

Constant: O(1)

Logarithmic: O(log n)

Linear: O(n)

Linearithmic: O(n log n)

Quadratic: O(n^2), etc.

**Analyze:**

**Loops: for and while loops**

**Recursive calls: use recurrence relations**

**3. Space Complexity**

How much extra memory is used?

Includes call stack for recursion, arrays, hashmaps, etc.

**4. Best, Worst, Average Case**

Example: QuickSort

Best: O(n log n)

Worst: O(n^2)

Average: O(n log n)

5. Trade-offs

Sometimes faster algorithms use more memory (space-time tradeoff).

✅ Example: Simple Design and Analysis

Problem: Find the maximum element in an array.

Pseudocode:

***function findMax(arr):***

***max = arr[0]***

***for i from 1 to length(arr) - 1:***

***if arr[i] > max:***

***max = arr[i]***

***return max***

Time Complexity: O(n) (one pass through the array)

Space Complexity: O(1) (only a variable max used)