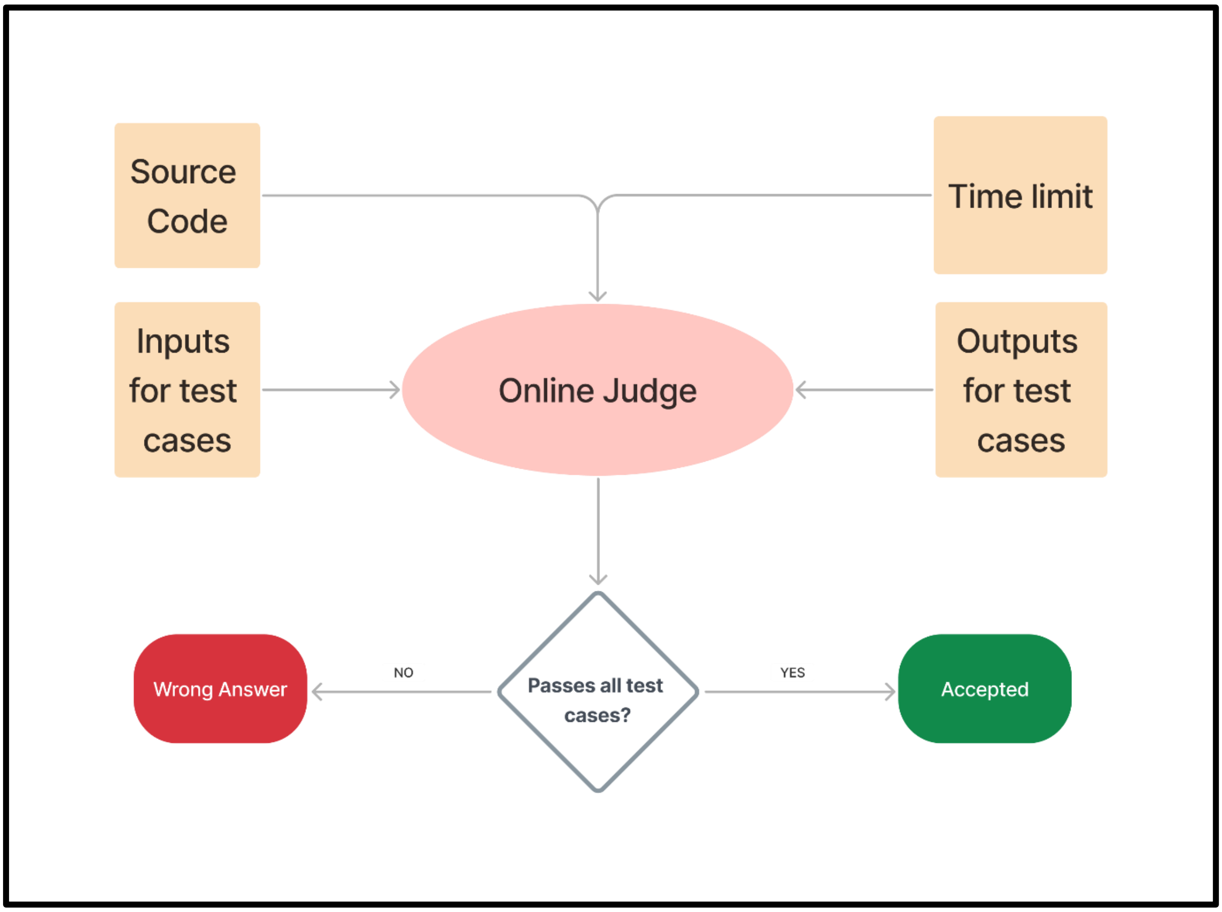
Online Judge:



Why is Complexity Analysis Important?

- To **predict performance** of algorithms.
- Helps in **choosing the best approach** for a problem.
- Avoids **wasting resources** like time and memory.

What is Complexity Analysis?

- It is a way to measure:
  - **Time Complexity**: How fast an algorithm runs.
  - **Space Complexity**: How much memory an algorithm uses.
- Gives an idea of how an algorithm will behave as the input size grows.

How to Analyze Complexity?

1. Time Complexity

- Measure the **number of steps** the algorithm takes.
- Focus on the **worst-case scenario** unless otherwise specified.
- Common examples:
    - Constant time: **O(1)** – No matter the input size, time stays the same.
    - Linear time: **O(n)** – Time grows with input size.
    - Quadratic time: **$O(n^2)$** – Time grows with the square of the input size.

2. Space Complexity

- Measure the **extra memory** used apart from the input.
- Includes:
    - Variables.
    - Data structures like arrays, stacks, or queues.
    - Function call stack.

Different Notations in Time Complexity

Key Idea

Time complexity measures how an algorithm's execution time grows with input size.

- **Better time complexity = faster code execution for large inputs.**

---

1. Asymptotic Notations

Used to describe algorithm performance:

1. **Big O (O)**:
    - Represents the **upper bound** (worst-case time).
    - Common time complexities:
        - **O (1)**: Constant time.
        - **O (log n)**: Logarithmic time.
        - **O(n)**: Linear time.
        - **O (n log n)**: Linear logarithmic time.
        - **$O(n^2)$**: Quadratic time.
        - **$O(2^n)$**: Exponential time.
2. **Omega (Ω)**:

    - Represents the **lower bound** (best-case time).

3. **Theta (θ)**:
    - Represents the **exact bound** (average-case time).

---

2. Practical Understanding

- **Big O** focuses on the worst case—important for scalability.
- **Omega** tells the fastest time possible.
- **Theta** ensures exact bounds (when growth is predictable).

---

3. Example

For an algorithm with **O(n):**

- Worst case: Linear growth with input size.
- It **cannot** take longer than quadratic growth (**$O(n^2)$**).
- It **cannot** run faster than logarithmic growth (**O(log n)**).

If it's **θ(n):**

- It always runs in **linear time**, with no deviations.

Summary

- **Big O** = Worst case.
- **Omega** = Best case.
- **Theta** = Exact bounds.

Understanding these helps evaluate algorithm efficiency for large inputs!