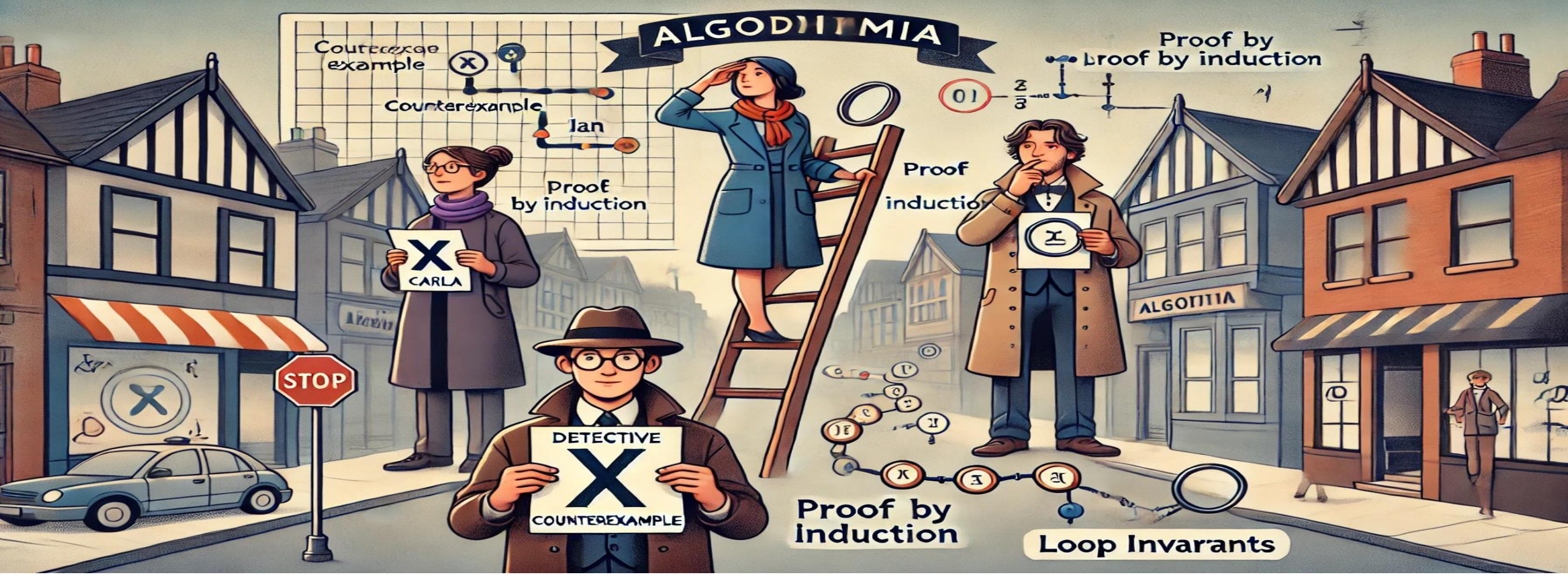
SCS1308 Foundations of Algorithms

Tutorial Session - 02

Correctness Of Algorithm



Let Me Tell You A Story.....

Solving Algorithm Correctness: A Mystery in Algorithmia

“In the bustling town of **Algorithmia**, a new sorting algorithm claims to be flawless. The town’s citizens are skeptical, and they’ve called in three expert detectives to uncover the truth. Each detective has their own unique approach to solving mysteries: **Carla** uses **counterexamples**, **Ian** applies **induction**, and **Lily** investigates **loop invariants**. Let's follow them as they uncover the truth! Each detective uses one of these methods to uncover the truth about the algorithm’s correctness. Let’s follow them on their journey! “

Detective Carla - The Counterexample Specialist

“Carla’s method is straightforward: She looks for flaws. If she finds even a single counterexample where the algorithm fails, the claim of correctness is disproven.”

What does Carla do?

- Finds specific inputs that break the algorithm.
- Proves the claim isn’t true in all cases.

Example:

- **Claim:** "Every integer is the sum of two squares."
- **Counterexample:** **03** is not the sum of any two squares.

Can you think of a situation where an algorithm might fail for a specific input?

Claim: "Every integer is the sum of two squares.“

A counterexample provides a specific integer that cannot be written as the sum of two squares. Let's consider the integer 3.

Testing the Claim for n=3:

- We need to check if there exist **integers** a and b such that:

$$a^2 + b^2 = 3$$

- Let’s try different values of a and b:

If **a=0** , then:

$$0^2 + b^2 = 3 \text{ ----} \rightarrow b^2 = 3$$

- But $b^2 = 3$ has no integer solution because 3 is not a perfect square.

If **a=1**, then:

$$1^2 + b^2 = 3 \text{ ----} \rightarrow b^2 = 3 - 1 = 2$$

Again, $b^2 = 3$ has no integer solution because 2 is not a perfect square.

“There are no integers a and b such that $a^2 + b^2 = 3$. Therefore, 3 is a counterexample, disproving the claim that "every integer is the sum of two squares."

Detective Ian - The Induction Expert

"Ian's approach is systematic. He proves correctness step by step."

What does Ian do?

- Proves the base case (n=1) is correct.
- Assumes correctness for n=k (inductive hypothesis).
- Proves correctness for n = k+1.

Example:

- *Claim*: The sum of the first n integers is $\frac{n(n+1)}{2}$.
- Ian shows it works for n=1 , assumes it works for n=k, and proves it for n=k+1.

Why is proving the base case essential before using induction?

The base case ensures that the statement being proved is true for the smallest or simplest instance, forming the **foundation** for the inductive argument.

For the formula **S(n)=** $\frac{n(n+1)}{2}$ (sum of first n integers):

base case (n=1) ensures the formula works for the simplest case:

S(n)= $\frac{1(1+1)}{2}$ **= 1**

Without proving this, the inductive step **(**$S(k+1)$**)** would have no foundation to build upon.

Inductive Hypothesis:

Assume the statement is true for n=k:

S(k)= $\frac{k(k+1)}{2}$

Inductive Step (n=k+1):

Add k+1 to both sides of the assumed formula:

S(K+1)= $S(K)+(K+1)$

Substitute **S(k)=** $\frac{k(k+1)}{2}$:

S(k+1)= $\frac{k(k+1)}{2}+(K+1)$

$\frac{K(K+1)+2(K+1)}{2} = \frac{(K+1)+(K+2)}{2}$

The statement holds for $n=k+1$. By induction, the formula is valid for all $n \geq 1$.

Detective Lily - The Loop Invariant Investigator

“Lily examines the process step by step. She ensures that the algorithm behaves correctly during every iteration of a loop.”

What does Lily do?

- Identifies a property (loop invariant) that must be true before and after every iteration.
- Proves the invariant holds during initialization, maintenance, and termination.

Example:

- *Claim*: A **linear search** correctly finds a value in an array.
- *Invariant*: At the start of each iteration, all checked elements do not contain the target value.

Can you think of a loop invariant for finding the maximum value in an array?

At the start of each iteration of the loop, the variable **max** contains the **largest value** among the elements examined so far.

Initialization:

- Before the loop starts, **max** is initialized to the first element of the array, $A[0]$.
- At this point, only one element has been considered, so the invariant holds.

Maintenance:

- During each iteration, the algorithm compares the **current element** $A[i]$ with **max**.
- If $A[i]>max$, **max** is updated to $A[i]$;
- otherwise, max remains unchanged.
- After the iteration, **max** still contains the **largest value** among the elements checked so far.

Termination:

- When the loop ends, all elements of the array have been checked.
- By the invariant, **max** contains the **largest value** in the entire array.

Questions for Recap…….

1. What are the three key properties that a good loop invariant must satisfy, and how do they help prove the correctness of an algorithm?
2. What are the three main steps in a proof by induction, and how does each step contribute to demonstrating that the statement is true for all natural numbers?