

# Evaluating Algorithms

---

## Textbook

---

# Evaluating Algorithms

When we create a set of instructions, or an **algorithm**, we check how well it works by looking at three main things: if it's right, if it's easy to understand, and if it's quick.

## Correctness

Being correct means the algorithm always gives the right answer for any proper input you give it, and it finishes its job completely. If an algorithm isn't correct, nothing else about it really matters.

Here's how to check if an algorithm is correct:

- **Test with Many Inputs:** Try running your algorithm with different kinds of data:
  - **Normal Situations:** Use inputs that are typical and what you'd usually expect.
  - **Extreme Situations:** Use unusual or boundary inputs, like empty lists, lists with only one item, or lists where all the items are the same.
  - **Bad Inputs:** See what happens if the input isn't what the algorithm expects. A good algorithm should handle these situations smoothly, perhaps by showing an error message.
- **Debugging:** This is when you find and fix problems, or "bugs," in your algorithm that stop it from working right.
  - **Go Step-by-Step:** Imagine you're the computer and manually go through each step of the algorithm with a specific input.
  - **Show Values:** In programming, you can temporarily display the values of different parts of your code at various points to see what's happening internally.

Example: Finding the Smallest Number

If your algorithm is supposed to find the smallest number in a list like [5,2,8,1,9], it should correctly give you 1. If the list is empty, like [], it should manage that situation properly, perhaps by telling you there's an error.

## Clarity: Is it easy to understand?

Clarity, also known as readability, refers to how simple it is for a person to understand your algorithm. This is super important for fixing problems, working with others, and making updates in the future.

Here's how to make an algorithm clear:

- **Good Names:** Use names for parts of your code that clearly describe what they are (for instance, use `student_score` instead of just `s`).

- **Notes:** Add comments to explain complicated sections or choices you made when designing the algorithm.
- **Organized Steps:** Make sure the algorithm's flow is logical and simple to follow.

## Calculating an Average

Let's look at a common algorithm used to find the average of a group of numbers and examine it.

- **Is it correct?** Does it do what it's supposed to? Try this algorithm with different inputs and see if it gives the right answer. Does it have any errors?
- **Is it clear?** Does it make sense to someone looking at it?

### Less Clear Algorithm:

```
1 function calc(data):
2     total = 0
3     i = 0
4     while i < length(data):
5         total = total + data[i]
6         i = i + 1
7     if length(data) > 0:
8         return total / length(data)
9     else:
10        return 0
11
```

### More Clear Algorithm:

```
1 function calculate_average(numbers_list):
2     if length(numbers_list) == 0:
3         return 0 # Handle empty list
4
5     sum_of_numbers = 0
6     for number in numbers_list:
7         sum_of_numbers = sum_of_numbers + number
8
9     average_result = sum_of_numbers / length(numbers_list)
10    return average_result
11
```

The second version is easier to understand because it uses better names and simpler logic.

## Invitation

Take the algorithm above and try to improve it! What changes can you make? How could you write a new program that does the same job or does it even better?

Software testing is vital for **complex software systems**. Various types of testing ensure quality.

# Types of Software Testing

- **Unit Testing:** Checks individual code pieces (e.g., a single function).
- **Integration Testing:** Verifies if different software modules work together.
- **System Testing:** Tests the complete system against all requirements.
- **Regression Testing:** Reruns old tests after changes to ensure nothing broke.
- **Performance Testing:** Assesses speed and stability under various loads, including **load** and **stress testing**.
- **Security Testing:** Finds vulnerabilities to prevent breaches.
- **User Acceptance Testing (UAT):** End-users test the system to ensure it meets their needs.

## Automated Testing Platforms

**Automated testing platforms** are tools that run tests automatically, saving time and increasing accuracy, especially for complex systems.

- **Selenium:** For automating web application testing.
- **Appium:** For automating mobile application testing (iOS and Android).
- **Pytest** (Python) and **JUnit** (Java): Frameworks for unit and integration testing.
- Other platforms handle **performance testing** (e.g., JMeter) or **API testing** (e.g., Postman).

These platforms boost efficiency, accuracy, and test coverage, leading to significant long-term savings.

## Computationally Unsolvable Problems

It's important to know that some problems are **computationally unsolvable**, meaning no algorithm can find a correct solution for all inputs. The **Halting Problem** is a famous example: there's no program that can always tell if any other given program will ever stop running. Understanding these limits helps developers focus on solvable problems and design systems that handle unsolvable situations gracefully.

## Critical Thinking Questions

1. Imagine you're creating an algorithm for a system that controls stoplights at a busy intersection. Why would it be much more important to thoroughly test this algorithm with "extreme situations" (like an unusually large amount of traffic going one way, or a complete power failure) rather than just with typical traffic flow?
2. You've been given two different algorithms that both correctly put a list of names in alphabetical order. One algorithm is much longer and uses very short, confusing names for its parts, while the other is shorter and uses descriptive names and includes notes. If you had to choose one to maintain and update for the next five years, which one would you pick and why?
3. An algorithm designed to figure out student grades works perfectly for students who turn in all their assignments. However, when it's used for a student who has missing assignments, the program stops working. Which part of evaluating an algorithm (correctness, clarity, or efficiency) does this problem mostly relate to, and why is fixing this specific issue crucial before the algorithm can be trusted?

## Questions (5)

**1. You create an algorithm that is supposed to find the largest number in a list. When you give it the list [3, 7, 2, 9], it correctly outputs 9. What aspect of algorithm evaluation does this demonstrate?**

MULTIPLE CHOICE

**Choose the correct answer:**

- A. Clarity
- B. Efficiency
- C. Correctness
- D. Debugging

**2. Your algorithm is designed to calculate an average. When you give it an empty list [], it produces an error message instead of crashing. This shows the algorithm handles what well?**

MULTIPLE CHOICE

**Choose the correct answer:**

- A. Typical Cases
- B. Clarity
- C. Edge Cases
- D. Efficiency

**3. You are looking at two algorithms. One uses variable names like x, y, z, while the other uses names like student\_name, grade\_average. Which aspect of algorithm evaluation is better in the second example?**

MULTIPLE CHOICE

**Choose the correct answer:**

- A. Correctness
- B. Efficiency
- C. Clarity
- D. Debugging

**4. Why are "comments" important in an algorithm, even if the algorithm works perfectly without them?**

MULTIPLE CHOICE

**Choose the correct answer:**

- A. They make the algorithm run faster.
- B. They hide errors from the computer.
- C. They help humans understand complex parts or design choices.
- D. They are only for very simple algorithms.

**5. When you are trying to find and fix mistakes in your algorithm, what is this process called?**

MULTIPLE CHOICE

**Choose the correct answer:**

- A. Compiling
- B. Debugging
- C. Evaluating
- D. Commenting

## Answer Keys & Solutions

### Questions

1. You create an algorithm that is supposed to find the largest number in a list. When you give it the list [3, 7, 2, 9], it correctly outputs 9. What aspect of algorithm evaluation does this demonstrate?

MULTIPLE CHOICE

Correct Answer:

- |                |             |
|----------------|-------------|
| A. Clarity     | ✗ Incorrect |
| B. Efficiency  | ✗ Incorrect |
| C. Correctness | ✓ Correct   |
| D. Debugging   | ✗ Incorrect |

#### Explanation:

Think about whether the algorithm gives the right answer for a normal input.

2. Your algorithm is designed to calculate an average. When you give it an empty list [], it produces an error message instead of crashing. This shows the algorithm handles what well?

MULTIPLE CHOICE

Correct Answer:

- |                  |             |
|------------------|-------------|
| A. Typical Cases | ✗ Incorrect |
| B. Clarity       | ✗ Incorrect |
| C. Edge Cases    | ✓ Correct   |
| D. Efficiency    | ✗ Incorrect |

#### Explanation:

Consider how a robust algorithm deals with unusual or extreme inputs.

3. You are looking at two algorithms. One uses variable names like `x`, `y`, `z`, while the other uses names like `student_name`, `grade_average`. Which aspect of algorithm evaluation is better in the second example?

MULTIPLE CHOICE

Correct Answer:

- A. Correctness ✗ Incorrect
- B. Efficiency ✗ Incorrect
- C. Clarity ✓ Correct
- D. Debugging ✗ Incorrect

**Explanation:**

Think about what makes code easy for a human to understand.

4. Why are "comments" important in an algorithm, even if the algorithm works perfectly without them?

MULTIPLE CHOICE

Correct Answer:

- A. They make the algorithm run faster. ✗ Incorrect
- B. They hide errors from the computer. ✗ Incorrect
- C. They help humans understand complex parts or design choices. ✓ Correct
- D. They are only for very simple algorithms. ✗ Incorrect

**Explanation:**

Think about who benefits from notes within the code.

5. When you are trying to find and fix mistakes in your algorithm, what is this process called?

MULTIPLE CHOICE

Correct Answer:

- A. Compiling ✗ Incorrect
- B. Debugging ✓ Correct

C. Evaluating

✗ Incorrect

D. Commenting

✗ Incorrect

**Explanation:**

The word literally means removing "bugs."