Problems 1:

Here's the algorithm, broken down with clear steps and explanations, along with how it translates to code:

Algorithm :

Start

Initialize Counter: Set a counter variable (let's call it count) to 0.

Loop Start: Begin a loop that will repeat until count equals 5.

Set Name: Assign the string "Internship" to a variable named Name.

Print Name (First Time): Display the current value of Name (which is "Internship").

Modify Name: Concatenate the string "2025 " to the beginning of the Name variable.

Print Name (Modified): Display the updated value of Name (which is now "2025Internship").

Increment Counter: Increase the value of count by 1.

Loop Condition Check: If count is less than 5, go back to step 3 (Loop Start).

Stop

Explain

The algorithm uses a loop to repeat the process of setting, modifying, and printing the Name variable.The count variable keeps track of how many times the loop has executed.The string 2025 is added to the beginning of the Name variable in each iteration, creating the desired output.

Example Python Code :

Python

```
count = 0


while count < 5:
    Name = "Internship"
    print(Name)
    Name = "2025 " + Name
    print(Name)
    count += 1
```

Explain the Code:

count = 0: Initializes the counter.

while count < 5:: Sets up the loop that repeats 5 times.

Name = "Internship": Sets the initial value of Name.

print(Name): Prints the initial value.

Name = "2025 " + Name: Modifies Name by adding "2025 " to the front.

print(Name): Prints the modified value.

count += 1: Increments the counter.

Problem 2:

.

Algorithm:

START

INITIALIZE count to 0.

LOOP START:

IF count is greater than or equal to 5, **GOTO** step 10 (STOP).

ASSIGN Name to the string "Internship".

OUTPUT the value of Name.

ASSIGN Name to the string "2025 " concatenated with the current value of Name.

OUTPUT the value of Name.

INCREMENT count by 1.

GOTO step 3 (LOOP START).

STOP

Key Improvements :

Explicit Conditional Jump: Step 3 now clearly states the condition for exiting the loop and the destination step.

Formalized Actions: Words like "ASSIGN," "OUTPUT," and "INCREMENT" are used to make the actions more explicit, which is common in formal algorithm descriptions.

Clear GOTO: The GOTO statement makes the loop flow very clear.

Problem 3:

You've presented two distinct algorithmic problems:

Finding the Maximum Number (Algorithm Provided):

Your provided algorithm is perfectly clear and correct. Here's a slightly more formalized version, emphasizing the key steps:

Algorithm: Finding the Maximum Number

START

INITIALIZE max to 0.

LOOP START:

IF there are no more numbers in the input, **GOTO** step 6 (PRINT).

INPUT a number.

COMPARISON:

IF number is greater than max, **ASSIGN** max to number.

GOTO step 3 (LOOP START).

PRINT the value of max.

STOP

Towers of Hanoi (Algorithm):

The Towers of Hanoi problem is a classic example of a recursive algorithm. Here's a clear breakdown:

Algorithm: Towers of Hanoi

Recursive Function: moveTower(n, source, destination, auxiliary)

Input:

n: The number of disks to move.

source: The needle where the disks are initially located.

destination: The needle where the disks need to be moved to.

auxiliary: The remaining needle used as an intermediary.

BASE CASE:

IF n is equal to 1, **THEN:**

MOVE the top disk from source to destination.

RETURN.

RECURSIVE STEP:

CALL moveTower(n - 1, source, auxiliary, destination). (Move n-1 disks from source to auxiliary using destination as the intermediary)

MOVE the nth disk (the largest disk) from source to destination.

CALL moveTower(n - 1, auxiliary, destination, source). (Move n-1 disksfrom auxiliary to destination using source as the intermediary)

RETURN.

Explanation of Towers of Hanoi Algorithm:

Base Case: When there's only one disk (n = 1), the solution is trivial: move it directly from the source to the destination.

Recursive Step: For n disks:

Move the top n - 1 disks from the source to the auxiliary needle, using the destination needle as the intermediary.

Move the largest disk (the nth disk) from the source to the destination needle.

Move the n - 1 disks from the auxiliary needle to the destination needle, using the source needle as the intermediary.

Recursive Calls: The algorithm calls itself to solve smaller subproblems, which is the essence of recursion.

64 Disks: To solve the original problem with 64 disks, you would call moveTower(64, "Source Needle", "Destination Needle", "Auxiliary Needle").

Important Note (Towers of Hanoi):

The Towers of Hanoi problem has an exponential time complexity (O(2^n)). This means that the number of moves required grows very quickly as the number of disks increases. For 64 disks, the number of moves is enormous, which is why the legend suggests it will take a very long time.

Problem 5:

START

INPUT num1 (4-digit number)

INPUT num2 (4-digit number)

INITIALIZE product to 0

INITIALIZE counter to 0

LOOP START:

IF counter is equal to num2, **GOTO** step 9 (PRINT).

ADD num1 to product.

INCREMENT counter by 1.

GOTO step 6 (LOOP START).

PRINT product

STOP

Explanation:

The algorithm simulates multiplication by repeated addition.

num1 is the multiplicand (the number being multiplied).

num2 is the multiplier (the number of times num1 is added).

product stores the result of the multiplication.

counter keeps track of how many times num1 has been added.

The while loop continues until counter reaches num2, at which point the loop terminates, and the product is printed.

Because num1 and num2 are assumed to be 4 digit numbers, the algorithm will correctly multiply them.

Problem 6

Method: Euclidean Algorithm (Iterative)

START

INPUT num1 (a non-prime number)

INPUT num2 (a non-prime number)

CHECK BASE CASES:

IF num1 is 0, THEN:

ASSIGN gcd to num2

GOTO step 8 (PRINT)

IF num2 is 0, THEN:

ASSIGN gcd to num1

GOTO step 8 (PRINT)

LOOP START (Euclidean Algorithm):

IF num2 is 0, THEN:

ASSIGN gcd to num1

GOTO step 8 (PRINT)

ASSIGN remainder to num1 modulo num2 (remainder = num1 % num2).

ASSIGN num1 to num2.

ASSIGN num2 to remainder.

GOTO step 5 (LOOP START).

ASSIGN gcd to num1 (This step is technically redundant, as the loop alreadyassigns it).

GOTO step 8 (PRINT).

PRINT gcd

STOP

Explanation:

Input: The algorithm takes two non-prime numbers as input.

Base Cases: It handles the cases where either number is 0, as the GCD is then the other number.

Euclidean Algorithm:

- The core of the algorithm is the Euclidean algorithm, which efficiently finds the GCD.
- It repeatedly replaces the larger number with the remainder of the division of the larger number by the smaller number.
- The process continues until the remainder is 0.
- The last non-zero remainder (or the value of num1 when num2 is 0) is the GCD.

Non-Prime Numbers: The Euclidean algorithm works correctly for any two integers, including non-prime numbers. The non-prime status of the numbers doesn't require any special handling.

Modulo Operator: The modulo operator (%) calculates the remainder of a division.

Output: The algorithm outputs the calculated GCD.