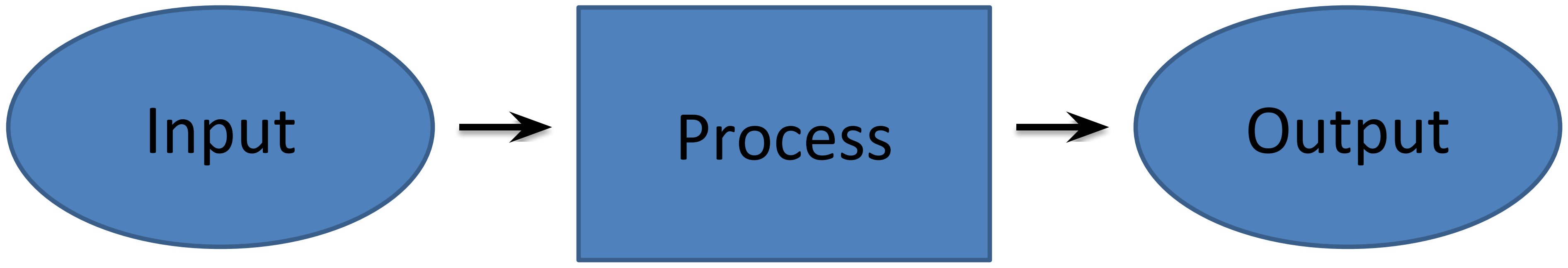
PROBLEM SOLVING THROUGH PYTHON PROGRAMMING

191GES102T

By,
Mrs. R. Poorni,
Assistant Professor / CSE

PROBLEM SOLVING



PYTHON

- Python is an interpreted, high-level, object-oriented programming language.
- Python was created in the late 1980s, and first released in 1991, by Guido van Rossum.
- Python 2.0, released in 2000, introduced new features and was discontinued.
- Later Python 3.0, released in 2008, was a major revision of the language.
- Python interpreters are available for many operating systems.

INTERPRETER	COMPILER
Interpreter translates just one statement of the program at a time into machine code.	Compiler scans the entire program and translates the whole of it into machine code at once.
An interpreter does not generate an intermediary code. Hence, an interpreter is highly efficient in terms of its memory.	A compiler always generates an intermediary object code. It will need further linking. Hence more memory is needed.

WHY PYTHON IS CALLED OBJECT ORIENTED

- Data controlling access to the code is called object oriented.
- It says what needs to be done rather than how it is done.
- Eg. Data stays at one place and the instruction moves around.
- Code controlling access to the data is called process oriented.
- Eg. Instruction stays at one place and the data moves around.

SYNTACTIC RULES FOR WRITING PYTHON

- Python is case sensitive.
- In python, there is no command terminator, which means no semicolon ; or anything.
- Code Indentation(tab): python doesn't use brackets. Python uses indentation for defining a block of code.
- In one line only a single executable statement should be written. To write two separate executable statements in a single line, you should use a semicolon ; to separate the commands.
- In python, you can use single quotes ‘ ‘, double quotes "" and even triple quotes ''' """ to represent string literals.
- you can write comments in your program using a # at the start. A comment is ignored while the python script is executed.

UNIT I
ALGORITHMIC PROBLEM SOLVING

Algorithms, building blocks of algorithms (statements, control flow, functions), notation (pseudo code, flow chart, programming language), algorithmic problem solving, simple strategies for developing algorithms (iteration, recursion). Case study: Towers of Hanoi, insertion sort, guess an integer number in a range.

PROBLEM SOLVING

- It is a systematic approach to define the problem and creates a number of solutions.

Techniques

- It helps in providing logic for solving a problem.
  - Algorithms
  - Flowcharts
  - Pseudocodes
  - Programs

STEPS IN PROBLEM SOLVING

- PROBLEM DEFINITION- define the problem statements and the requirements
- PROBLEM ANALYSIS- identify the problem inputs, check for the additional requirements
- ALGORITHM DEVELOPMENT- develop an algorithm that meets the requirements of solving the problem
- PROGRAM CODING- translating algorithm into syntax of a programming language.
- TESTING AND DEBUGGING- testing, running the program and executing the instructions. Debugging, correcting program code errors(incomplete knowledge)and mistakes(lack of attention)
- DOCUMENTATION- written manually what has been done in the overall problem

ALGORITHMS

- Algorithm is a step by step procedure.
- It defines a set of instructions to be executed in a certain order to get the desired output.
- It is independent of programming languages i.e the algorithm can be implemented in one or more programming languages.

CHARACTERISTICS OF AN ALGORITHM

- **Well-defined:** The instructions in the algorithm should be simple and well defined.
- **Unambiguous:** Algorithm should be clear and must lead to only one meaning.
- **Finiteness:** Algorithm must terminate after a finite number of steps.
- **Input:** The algorithm must receive an input.
- **Output:** After the algorithm gets terminated, the desired result must be obtained.
- **Independent:** The algorithm can be applied to various set of inputs.

EXAMPLE

Algorithm to add two numbers.

- Step 1: Start
- Step 2: Declare the variable a,b and c
- Step 3: read values for a and b
- Step 4: Add the values of a and b and store in c.
- Step 5: Write c
- Step 6: Stop

BUILDING BLOCKS OF ALGORITHM

- Any algorithm can be constructed using the following building blocks.
  a. Statement / Instruction
  b. State / Selection
  c. Control Flow
  d. Function

a. STATEMENT

- It describes the action and its sequence that the program carries.
- The action can be
  - Input data
  - Process data
  - Output processed data

b. STATE

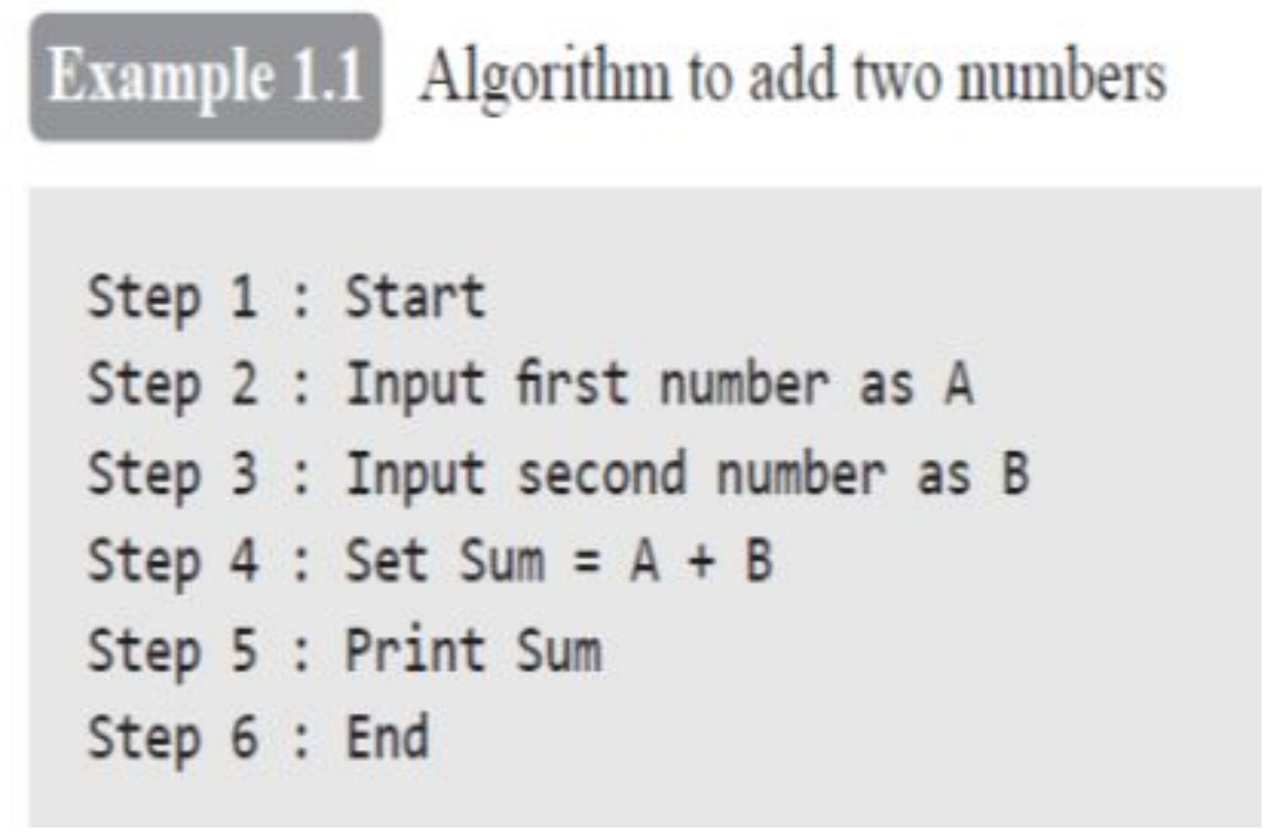
- It makes a transition or selection from one process to another under a specified condition.
- Eg:

```
If(a%2==0)
Print a is even
Or
Print a is odd
```

c. CONTROL FLOW

- The order of execution is called control flow.
- The control flow can be
    i. Sequence
    ii. Selection and
    iii. Iteration

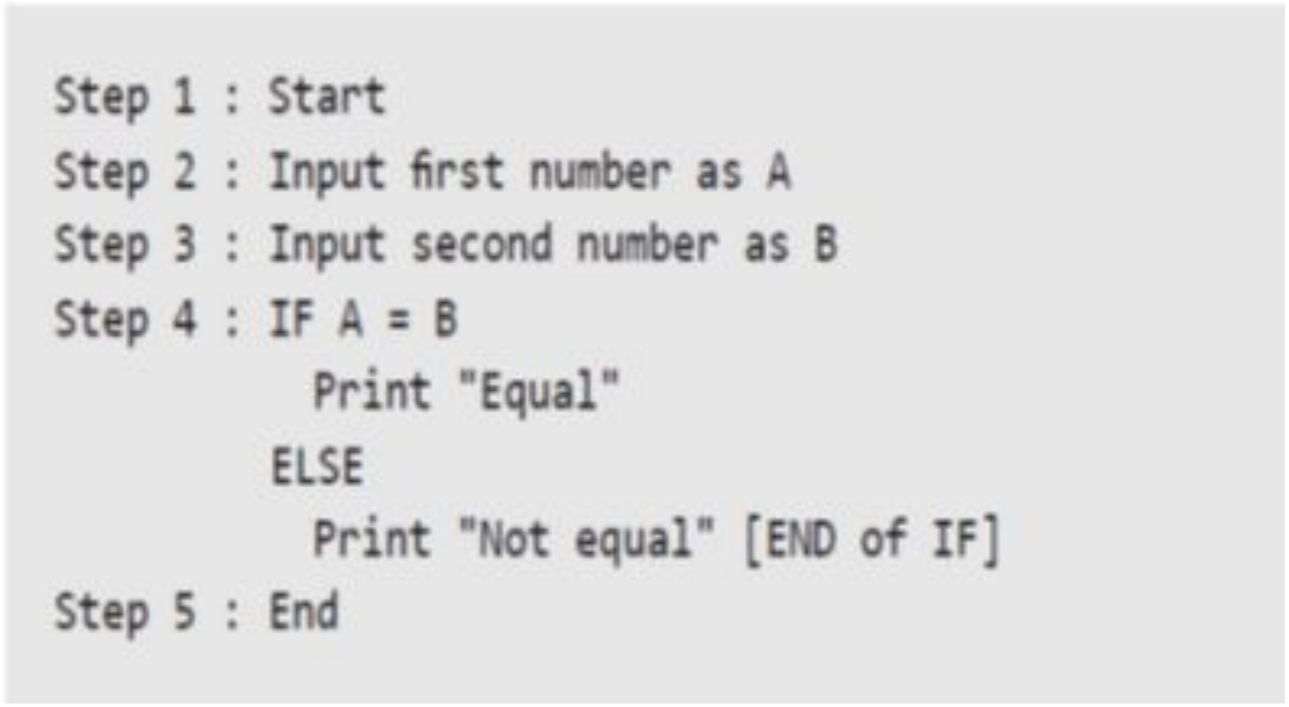
i. Sequence: Instructions are executed one after another. Eg. Addition of two numbers.



Example 1.1 Algorithm to add two numbers

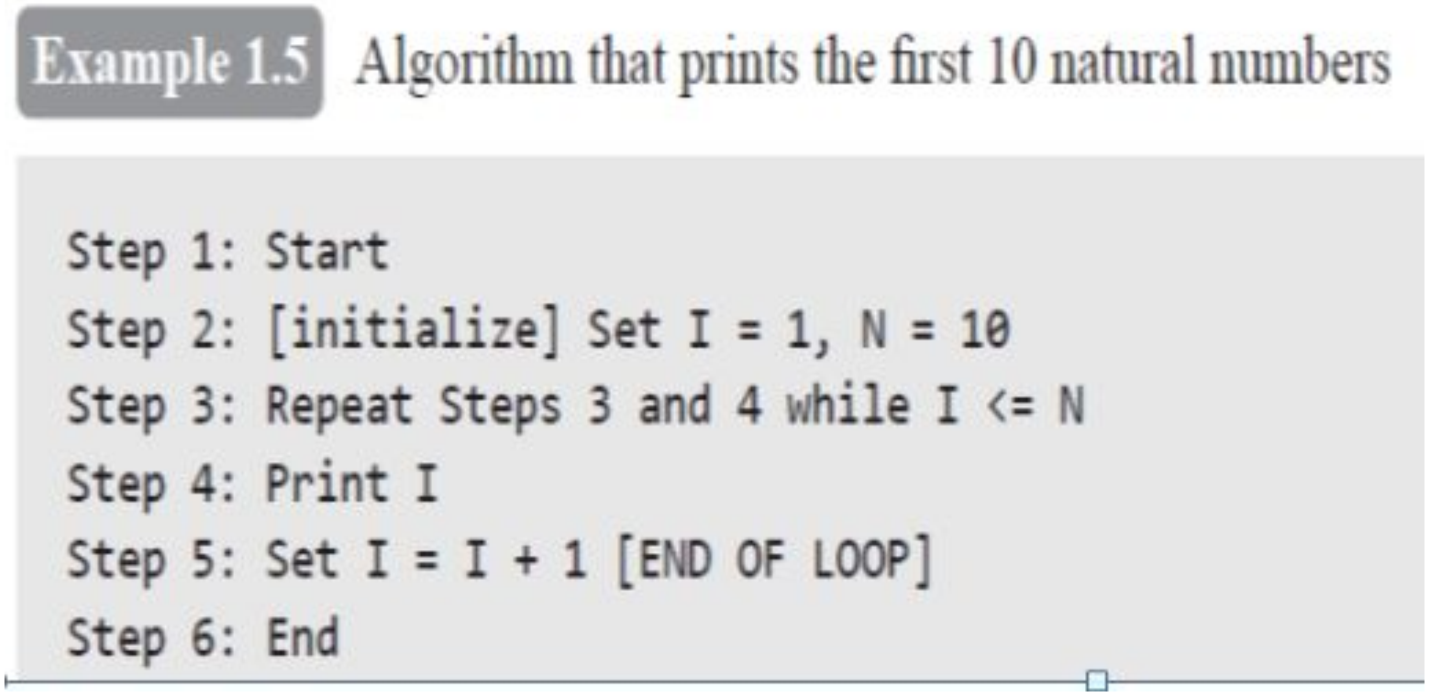
```
Step 1 : Start
Step 2 : Input first number as A
Step 3 : Input second number as B
Step 4 : Set Sum = A + B
Step 5 : Print Sum
Step 6 : End
```

ii. Selection: Control will be transferred to a part of a program based on the condition. The other part will not be executed or it remains untouched. Eg. Odd or Even.



```
Step 1 : Start
Step 2 : Input first number as A
Step 3 : Input second number as B
Step 4 : IF A = B
           Print "Equal"
         ELSE
           Print "Not equal" [END of IF]
Step 5 : End
```

iii. Iteration: It allows set of instructions to get executed repeatedly based on condition more than once.



Example 1.5 Algorithm that prints the first 10 natural numbers

```
Step 1: Start
Step 2: [initialize] Set I = 1, N = 10
Step 3: Repeat Steps 3 and 4 while I <= N
Step 4: Print I
Step 5: Set I = I + 1 [END OF LOOP]
Step 6: End
```

d. FUNCTIONS

- It allows a large task to divide into smaller blocks.
- Function is a sub program that consists of block of code.

Advantages

- – Line reduction
- – Code reuse
- – Data hiding
- – Easy understanding

$$\begin{tikzcd}
\textbf{Main Function} & \textbf{Sub function} \\
\text{Start} & \text{Read a,b \& c} \\
\text{Call add()} \arrow[ur] & \text{Add a and b and store in c} \\
\text{Stop} & \text{Print c} \\
 & \text{Return} \arrow[ul]
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Mul} \arrow[r, shift left] & \text{A*a} \arrow[l, shift left] \arrow[ddl, shift left] \\
\text{Square} & \\
\text{call Mul()} \arrow[uur, shift left] &
\end{tikzcd}$$

NOTATIONS

- The algorithm can be represented using
  - Pseudo code
  - Flow chart
  - Programming languages

PSEUDOCODE

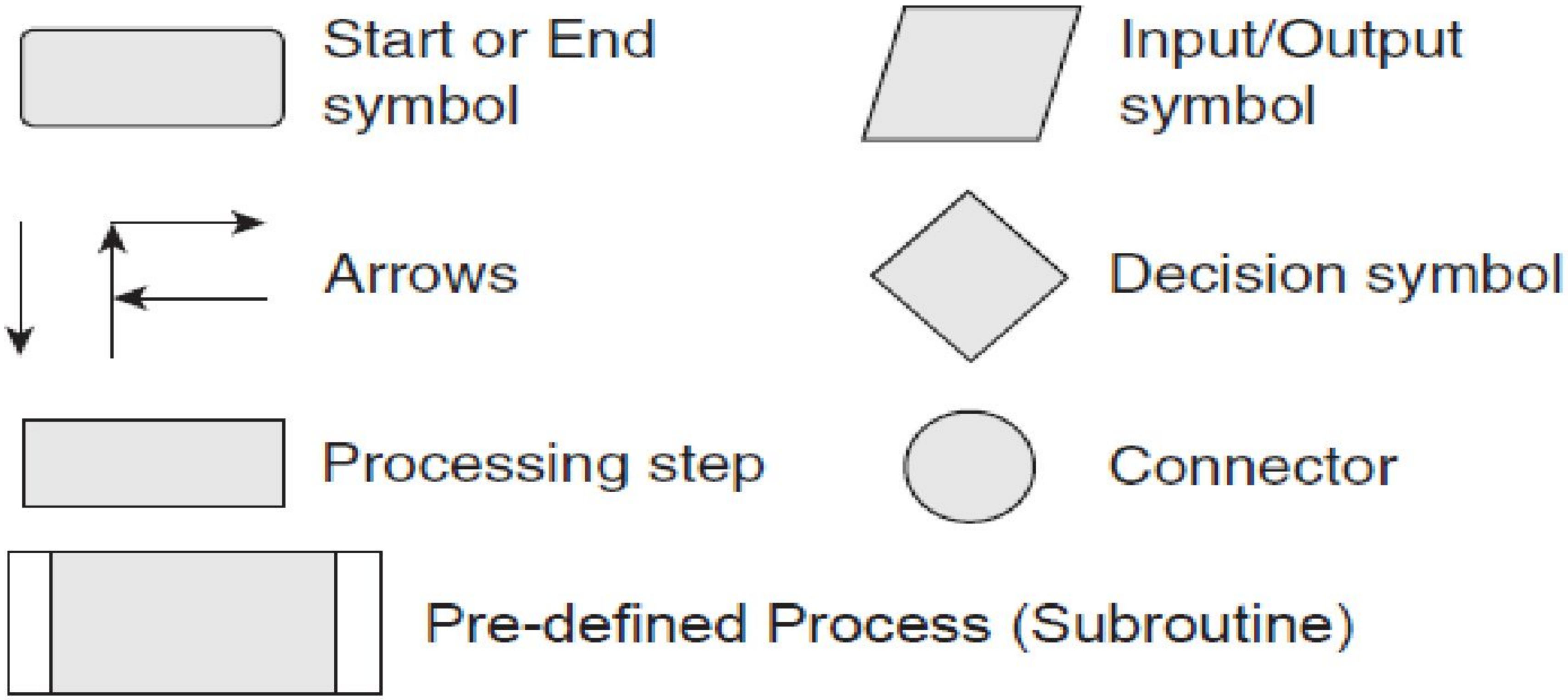
- Pseudocode is a compact and informal high-level description of an algorithm that uses the structural conventions of a programming language.
- It facilitates designers to focus on the logic of the algorithm without getting bogged down by the details of language syntax.
- It consist of short English phrases that explain specific tasks within a program’s algorithm. They should not include keywords in any specific computer language.
- The sole purpose of pseudocodes is to enhance human understandability of the solution.

PSEUDOCODE EXAMPLE

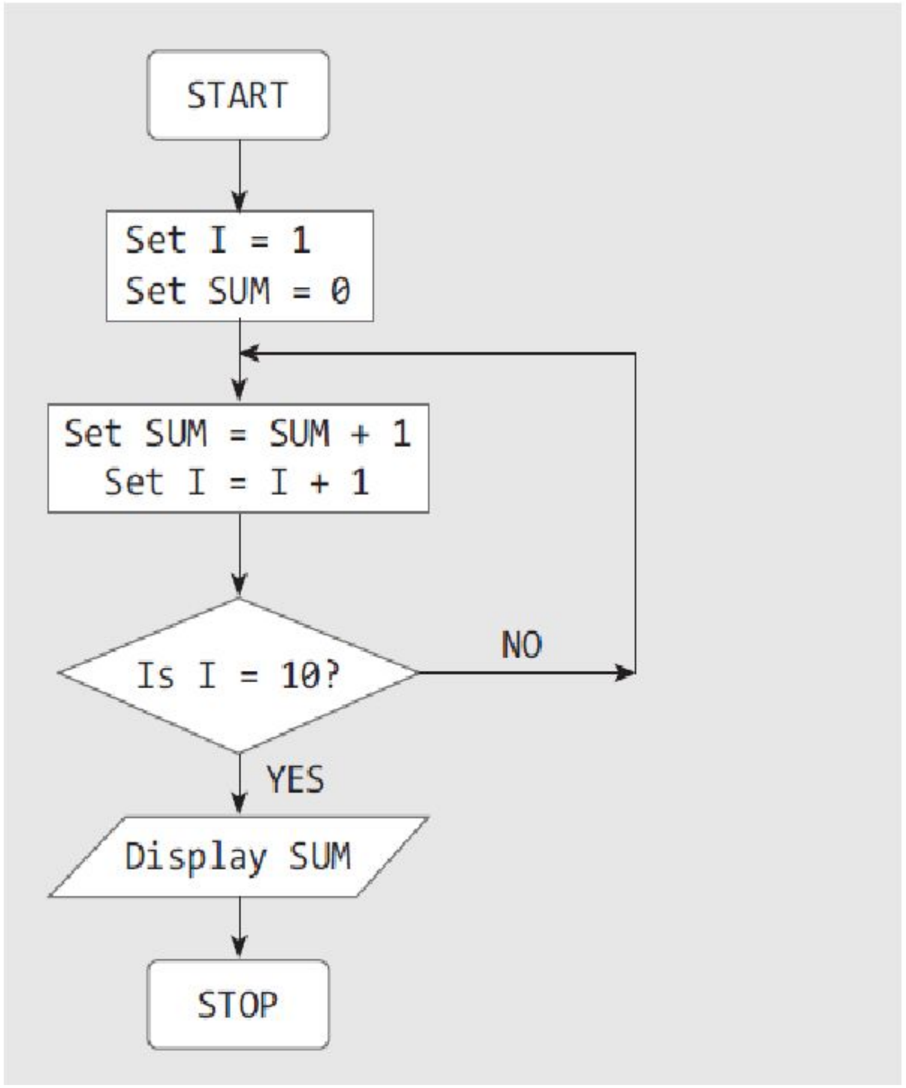
- BEGIN
- INPUT the first number
- INPUT the second number
- Add the two numbers (or) sum=num1+num2
- OUTPUT the sum
- END

FLOWCHARTS

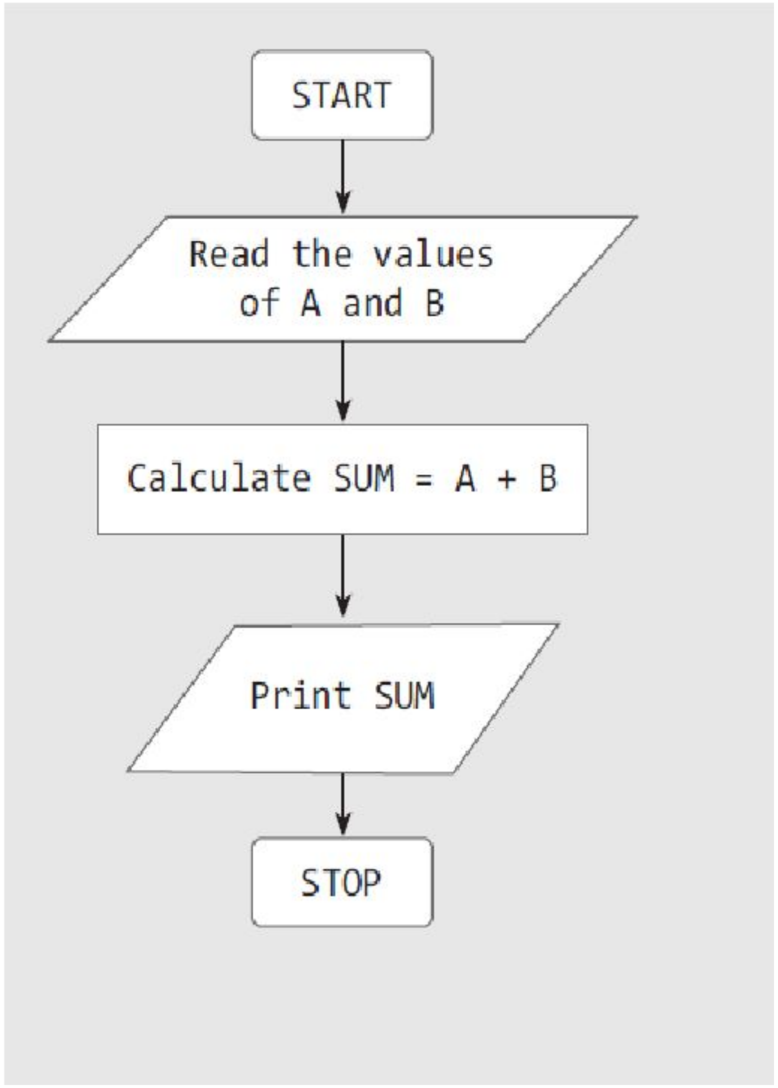
- A flowchart is a graphical or symbolic representation of a process.
- A flowchart is a diagrammatic representation that illustrates the sequence of steps that must be performed to solve a problem.
- It is usually drawn in the early stages of formulating computer solutions.
- Each step in the process is depicted by a different symbol and is associated with a short description.


Start or End symbol
Input/Output symbol
Arrows
Decision symbol
Processing step
Connector
Pre-defined Process (Subroutine)

Example 1.13 Draw a flowchart to calculate the sum of the first 10 natural numbers.



Example 1.14 Draw a flowchart to add tw numbers.



Advantages

- Communication is easy
- Effective analysis
- Documentation
- Easy to backtrack

Limitations

- Difficult to implement for large and complex programs
- Modifications may require redrawing of flowchart
- No well-defined standards.

PROGRAMMING LANGUAGES

- It is a set of symbols and rules for instructing a computer to perform specific tasks.
- The programmer must follow the rules to get the desired output.
- It is used to create programs that control the behaviour of the system
- Eg. C, C++, FORTRAN, Python
- Each language has unique syntax and semantics.

TYPES

- Machine language
- Low level or assembly level language
- High level language

- Irrespective of the language that a programmer uses, a program written using any programming language has to be converted into machine language so that the computer can understand it.
- This can be done using compiler or an interpreter.

ALGORITHMIC PROBLEM SOLVING

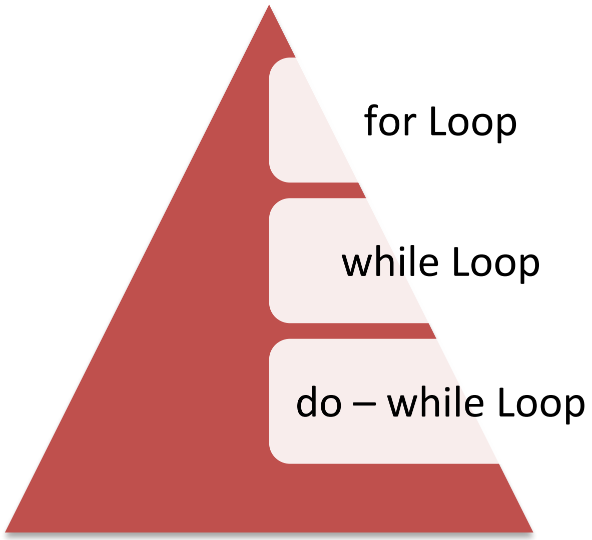
- Sequence of steps to be followed for designing an effective algorithm

1. [illegible]
2. [illegible]
3. [illegible]
4. [illegible]
5. [illegible]
6. [illegible]

STRATEGIES FOR DEVELOPING ALGORITHMS

- **Iteration** - Loops to repeat some part of code

```
1. Start
2. Set I=1, N=10
3. Repeat step 3 & 4 While I <=N
4. Print I
5. Set I = I+1     End of Loop
6. Stop
```

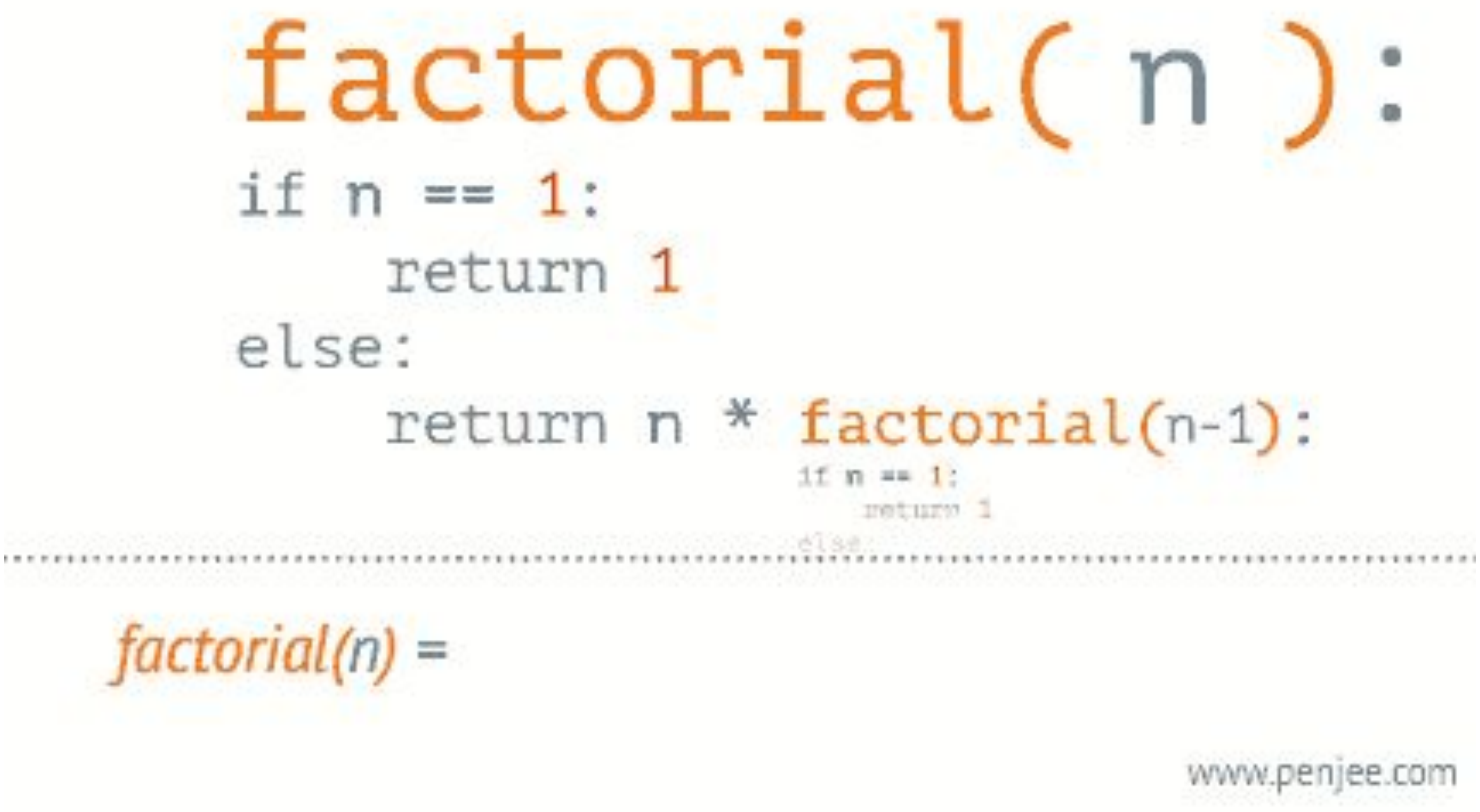


- **Recursion** - function call itself repeatedly until some specified condition has been satisfied.
- It solves a problem by breaking it into smaller sub programs.
- If n=1 ⍰ Base case, Otherwise recursive case.
- If n-1 cases can be solved the nth case can easily be solved.

Eg. Towers of Hanoi, Factorial

```
Factorial(n)
{
      if n==0 & n==1
                  return 1
      else
                  Print (   n * Factorial(n-1)   )
}
```

```
4!  =  4 * 3!
    =  4*3*2!
    =  4*3*2*1!
    =  4*3*2*1
    =   12
```



```
factorial( n ):
if n == 1:
    return 1
else:
    return n * factorial(n-1):
                 if n == 1:
                     return 1
                 else
```

factorial(n) =

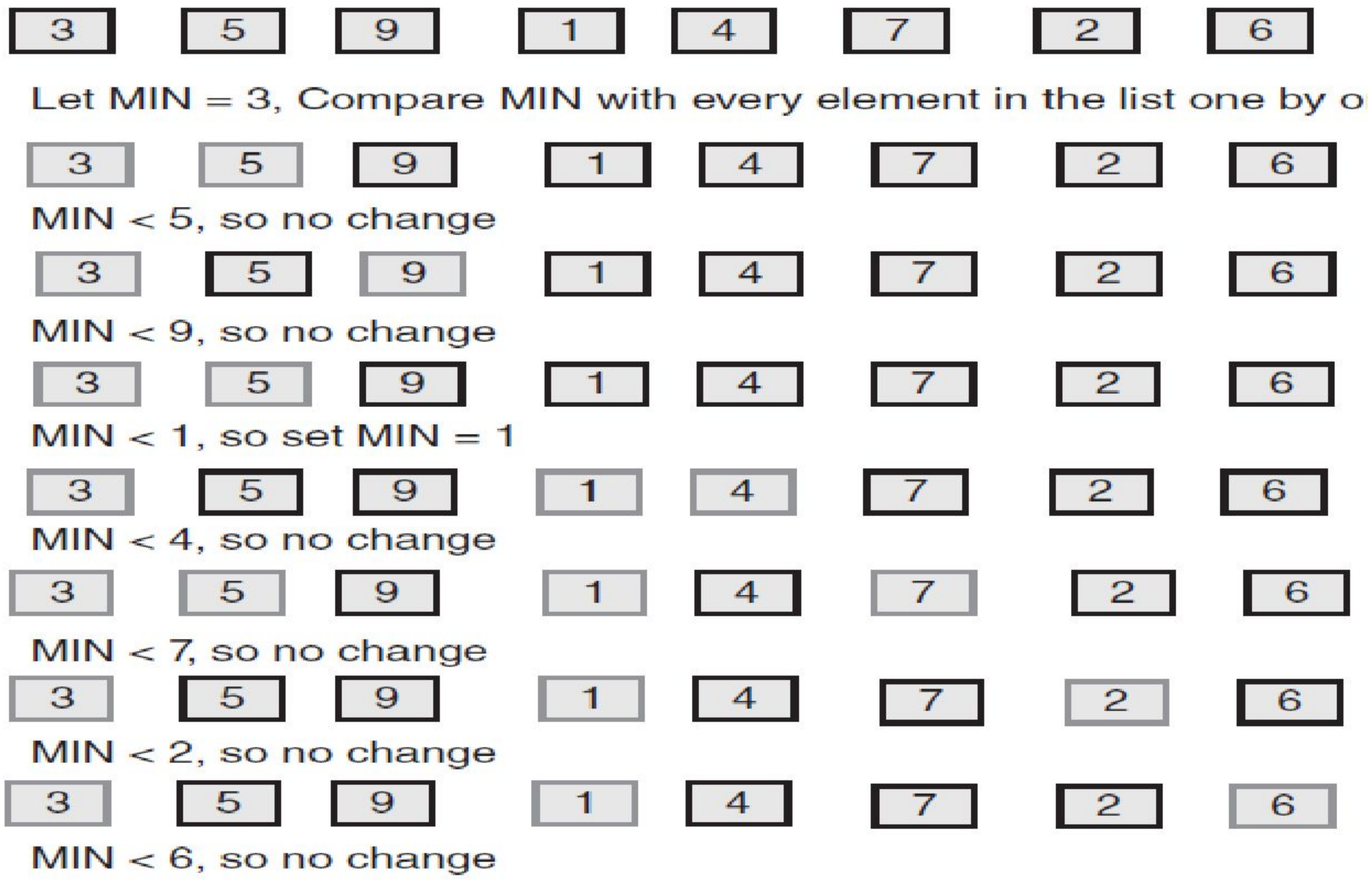
CASE STUDY

- Towers of Hanoi
- Insertion sort
- Guess an integer number in a range.

PROBLEM SOLVING FIND MINIMUM IN A LIST

- Set 1st number as minimum.
- Compare the minimum value with all the other numbers in the list.
  - If condition is true proceed with next element.
  - If condition fails then take that element as minimum element.
- Repeat step 2.
- Print minimum

Find Minimum in a List



Let MIN = 3, Compare MIN with every element in the list one by o

MIN < 5, so no change

MIN < 9, so no change

MIN < 1, so set MIN = 1

MIN < 4, so no change

MIN < 7, so no change

MIN < 2, so no change

MIN < 6, so no change

$\boxed{3}$ $\boxed{5}$ 9 1 4 7 2 6

min= 3

3<5, condition true, proceed with the next element.

step 2:

$\boxed{3}$ 5 $\boxed{9}$ 1 4 7 2 6

min= 3

min<9, condition true, proceed with the next element.

step 3:

$\boxed{3}$ 5 9 $\boxed{1}$ 4 7 2 6

min= 3

3<1, condition fails, change the min value as 1 (or) min=1

3 5 9 $\boxed{1}$ $\boxed{4}$ 7 2 6

min= 1

1<4, condition true, proceed with the next element.

3 5 9 $\boxed{1}$ 4 $\boxed{7}$ 2 6

1<7,

...

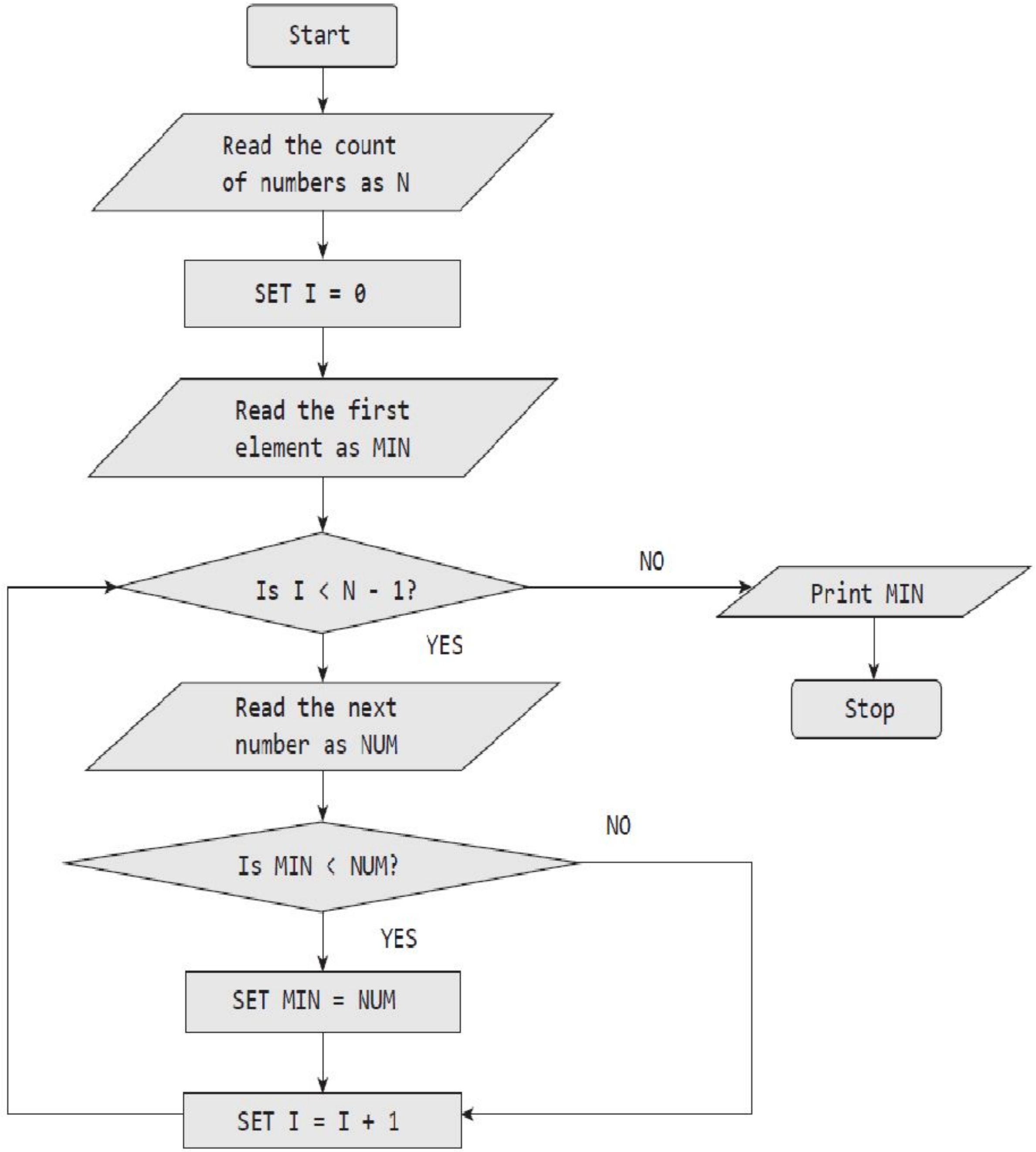
the minimum element is 1

Algorithm 1.1

```
Step 1:   Start
Step 2:   READ the count of numbers as N
Step 3:   SET I = 0
Step 4:   READ the first element as MIN
Step 5:   REPEAT Steps 6-8 WHILE I < N - 1
Step 6:   READ the next number as NUM
Step 7:   IF MIN < NUM
              SET MIN = NUM
Step 8:   SET I = I + 1
Step 9:   PRINT MIN
Step 10:  End
```

Pseudocode 1.1

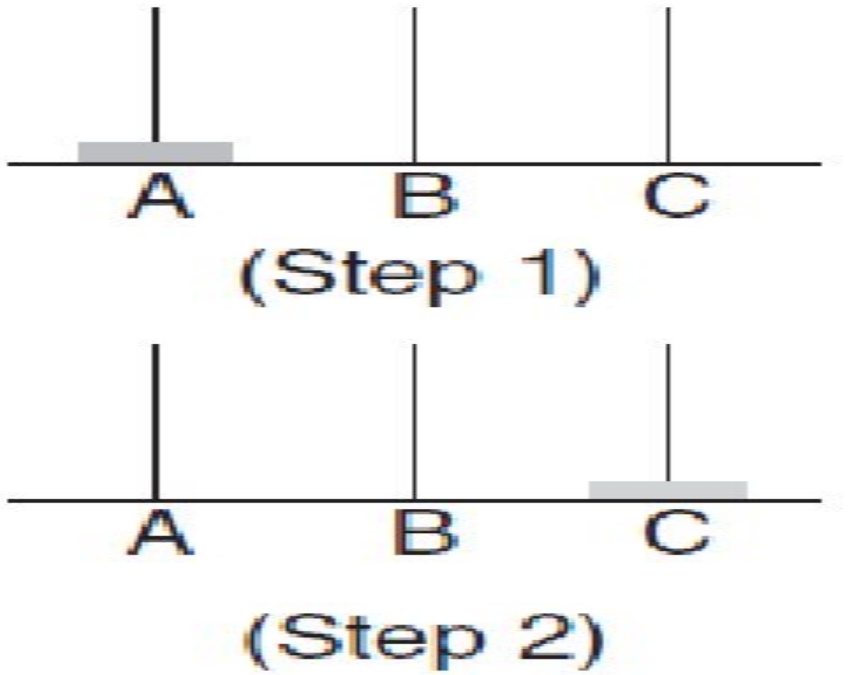
```
Read the count of numbers as N
Set I = 0
Read the first element as MIN
While I < N - 1
    Read the next number as NUM
        IF MIN < NUM
            Set MIN = NUM
        Set I = I + 1
Print MIN
End
Variables: I, N, NUM, MIN
```


Start
Read the count of numbers as N
SET I = 0
Read the first element as MIN
Is I < N - 1?
NO
Print MIN
Stop
YES
Read the next number as NUM
Is MIN < NUM?
NO
YES
SET MIN = NUM
SET I = I + 1

TOWER OF HANOI

- Only one disk can be moved at a time
- Only top disk can be removed
- No large disk can be placed over a small disk.

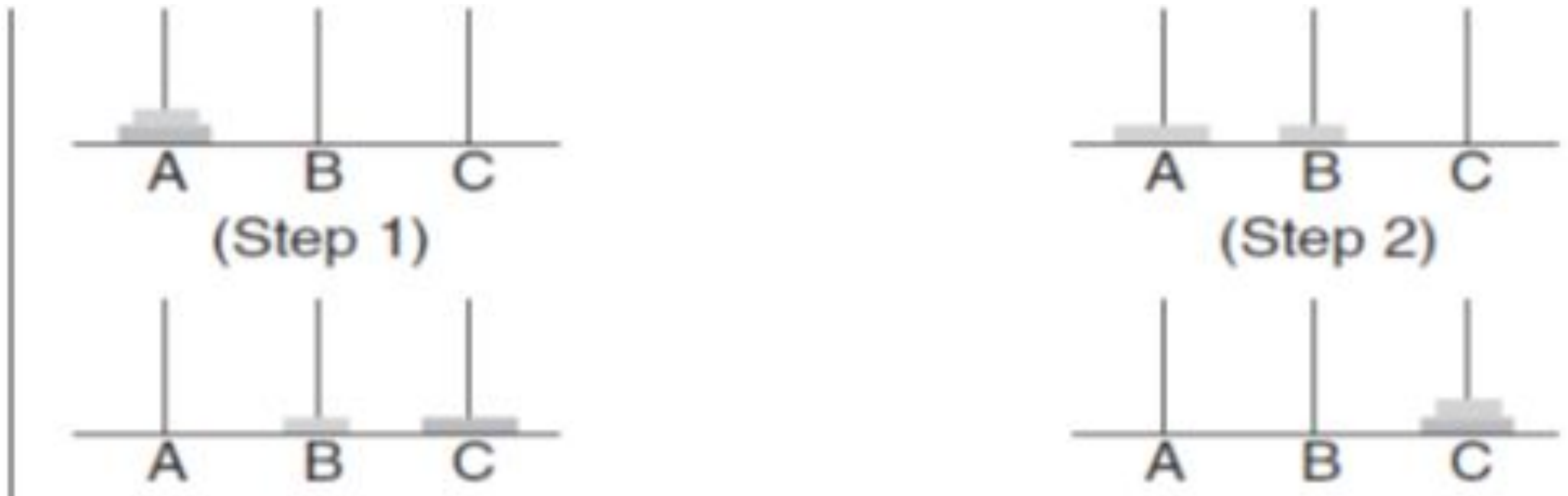
DISK = 1



(If there is only one ring, then simply move the ring from source to the destination.)

(a)

DISK = 2

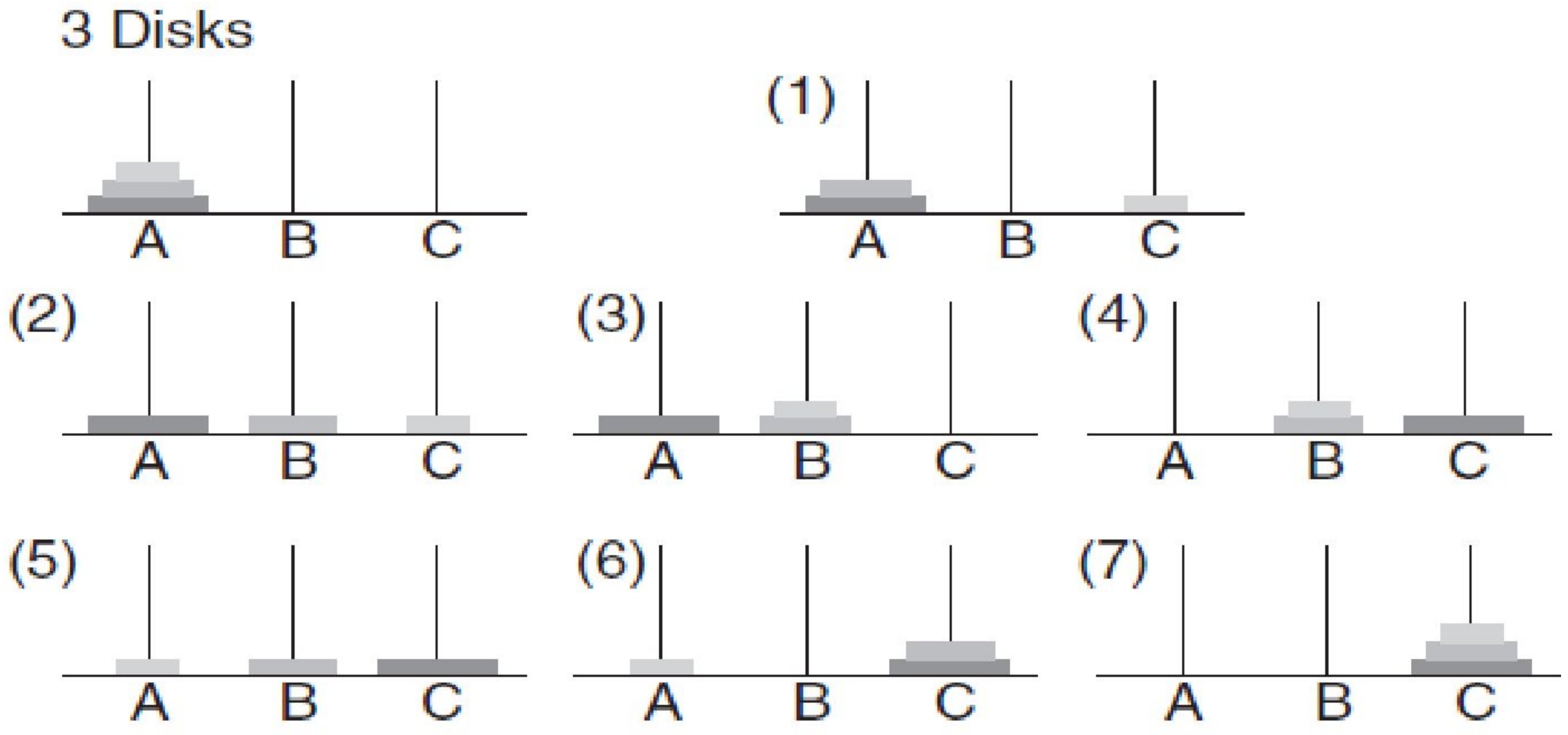


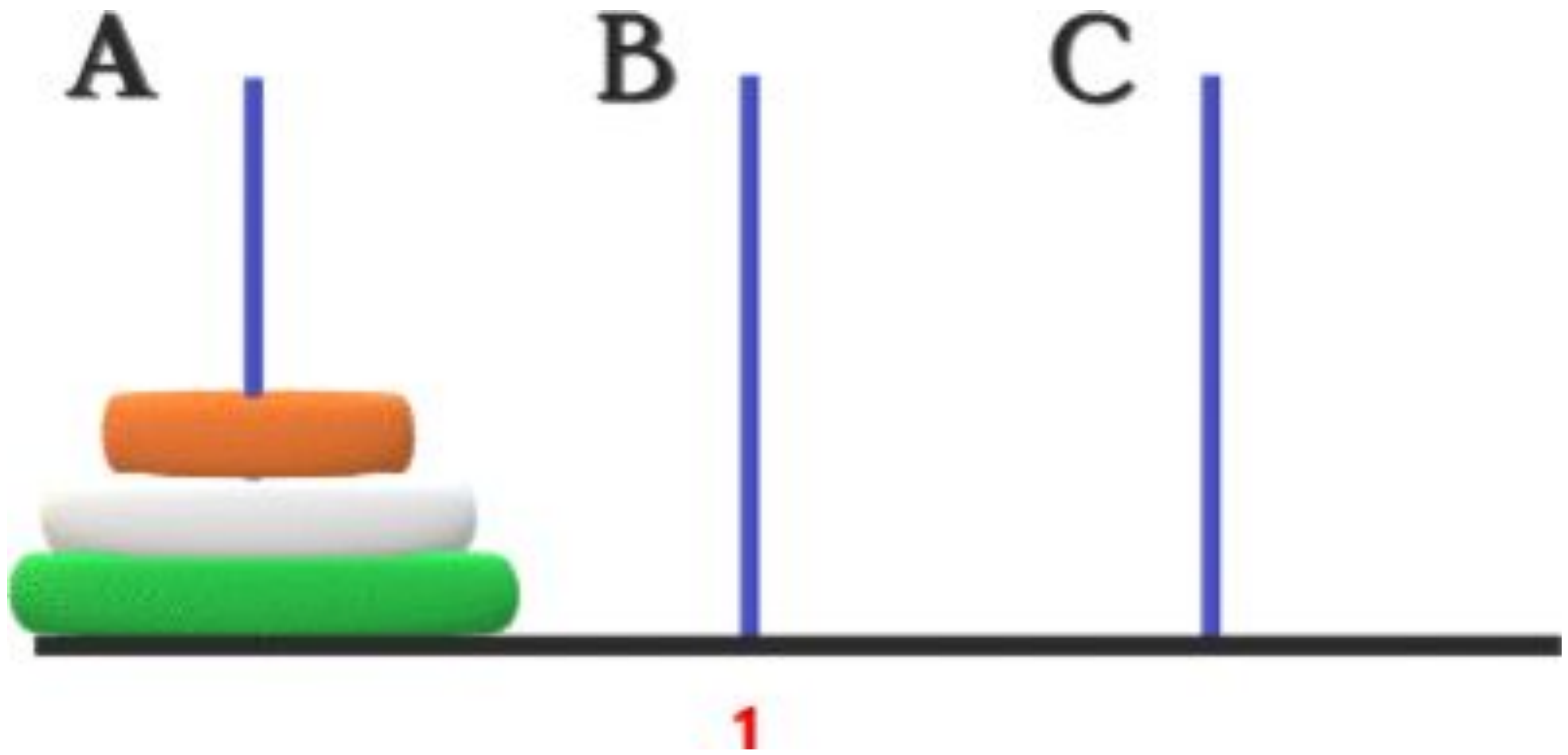
(Step 3) (Step 4)

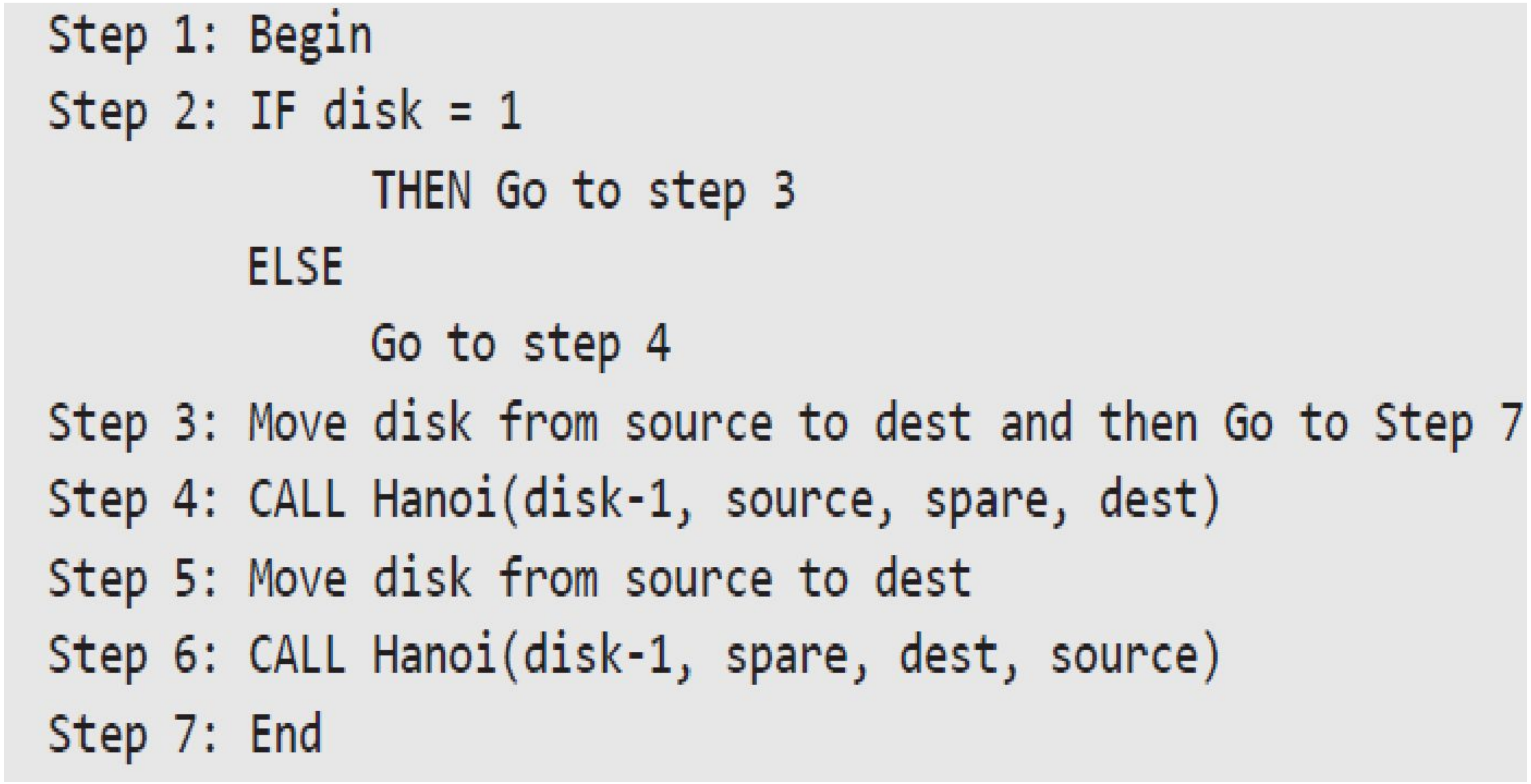
(If there is two rings, then first move ring 1 to the spare pole and then move ring 2 from source to the destination. Finally move ring 1 from spare to the destination.)

(b)

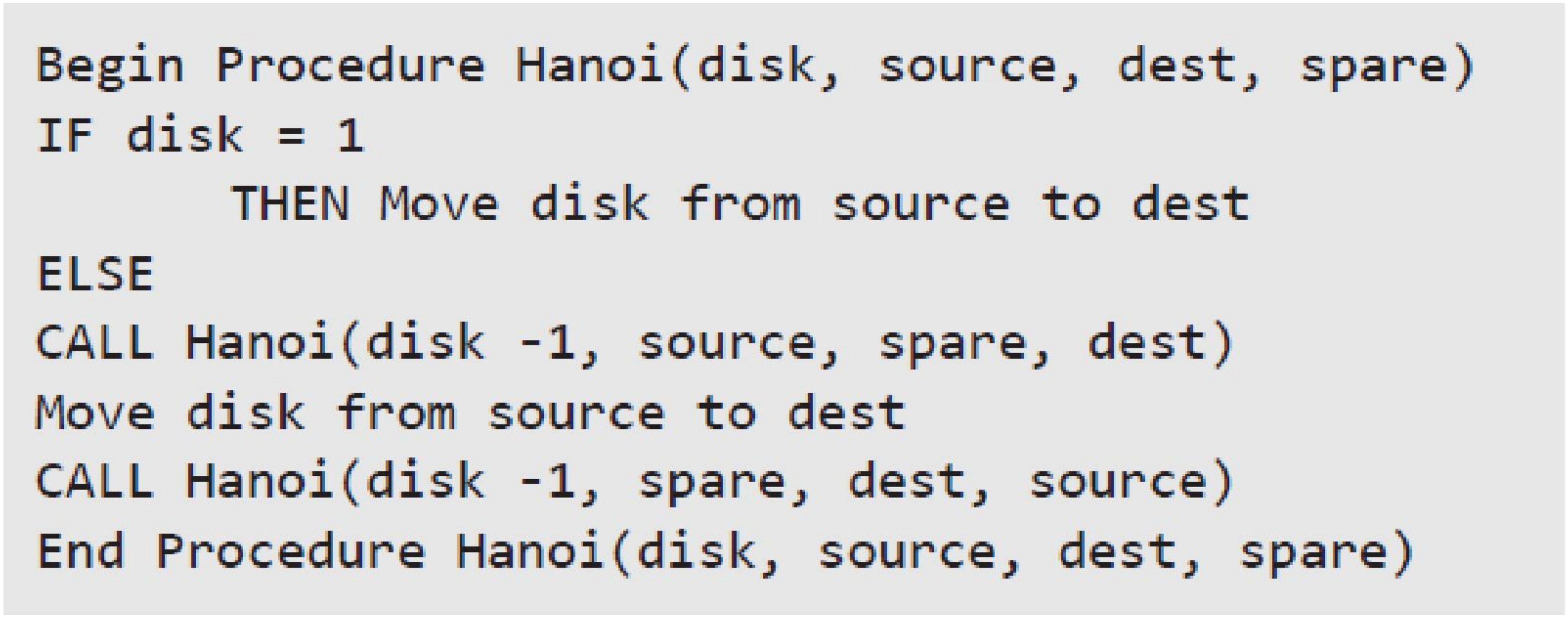
DISK = 3




A
B
C
1

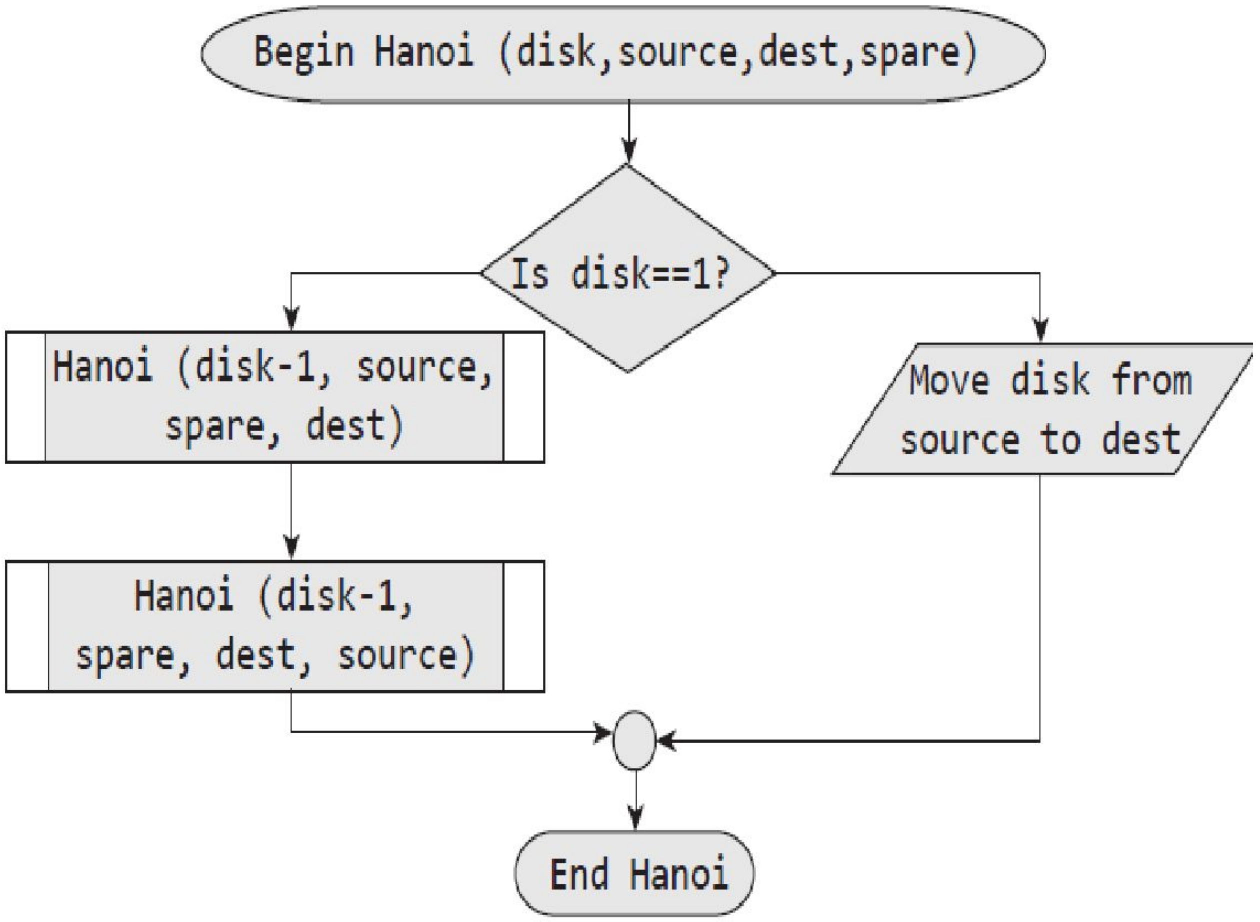


```
Step 1: Begin
Step 2: IF disk = 1
            THEN Go to step 3
        ELSE
            Go to step 4
Step 3: Move disk from source to dest and then Go to Step 7
Step 4: CALL Hanoi(disk-1, source, spare, dest)
Step 5: Move disk from source to dest
Step 6: CALL Hanoi(disk-1, spare, dest, source)
Step 7: End
```



```
Begin Procedure Hanoi(disk, source, dest, spare)
IF disk = 1
      THEN Move disk from source to dest
ELSE
CALL Hanoi(disk -1, source, spare, dest)
Move disk from source to dest
CALL Hanoi(disk -1, spare, dest, source)
End Procedure Hanoi(disk, source, dest, spare)
```

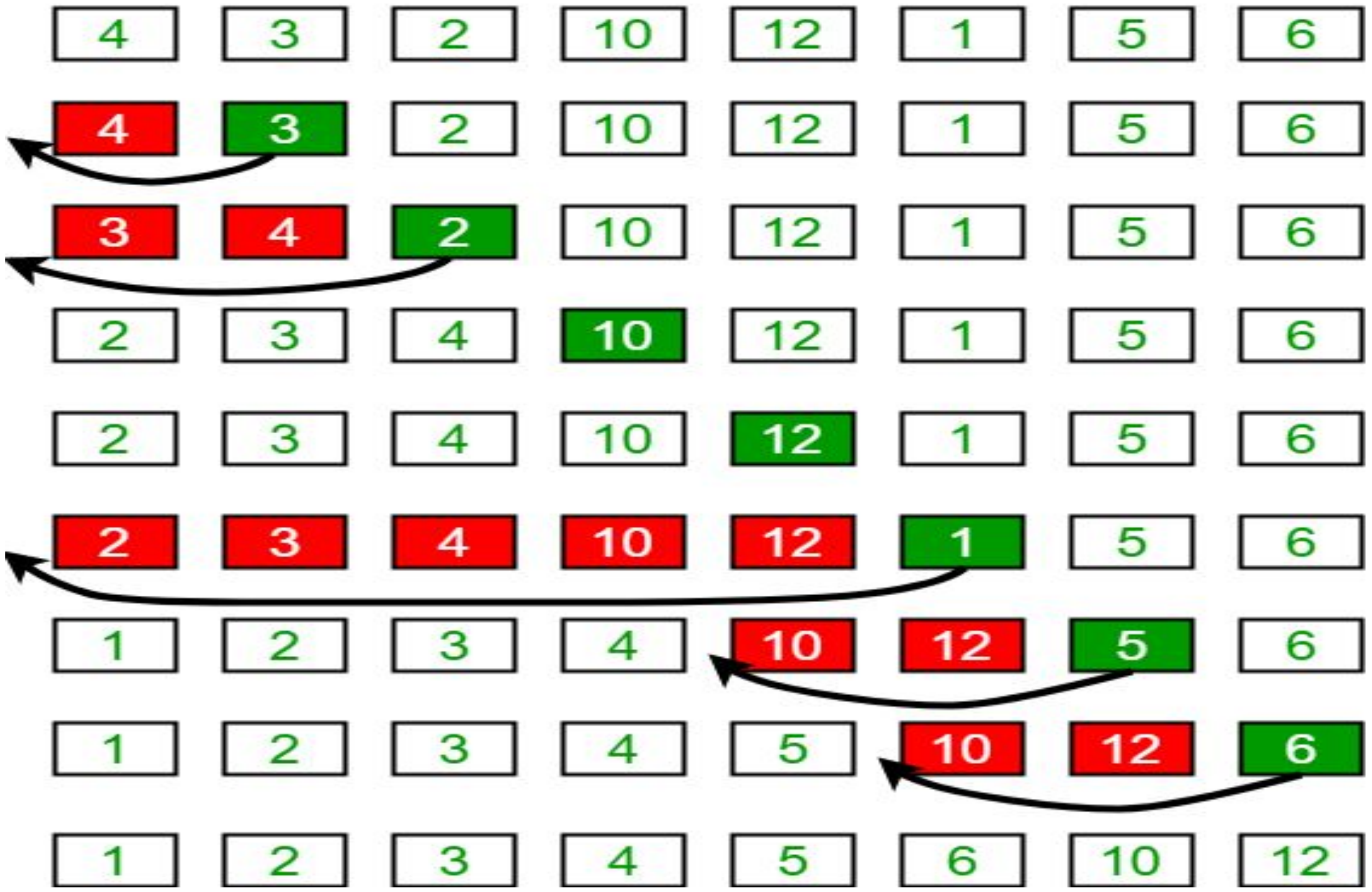
Tower of Hanoi


Begin Hanoi (disk,source,dest,spare)
Is disk==1?
Hanoi (disk-1, source, spare, dest)
Move disk from source to dest
Hanoi (disk-1, spare, dest, source)
End Hanoi

INSERTION SORT

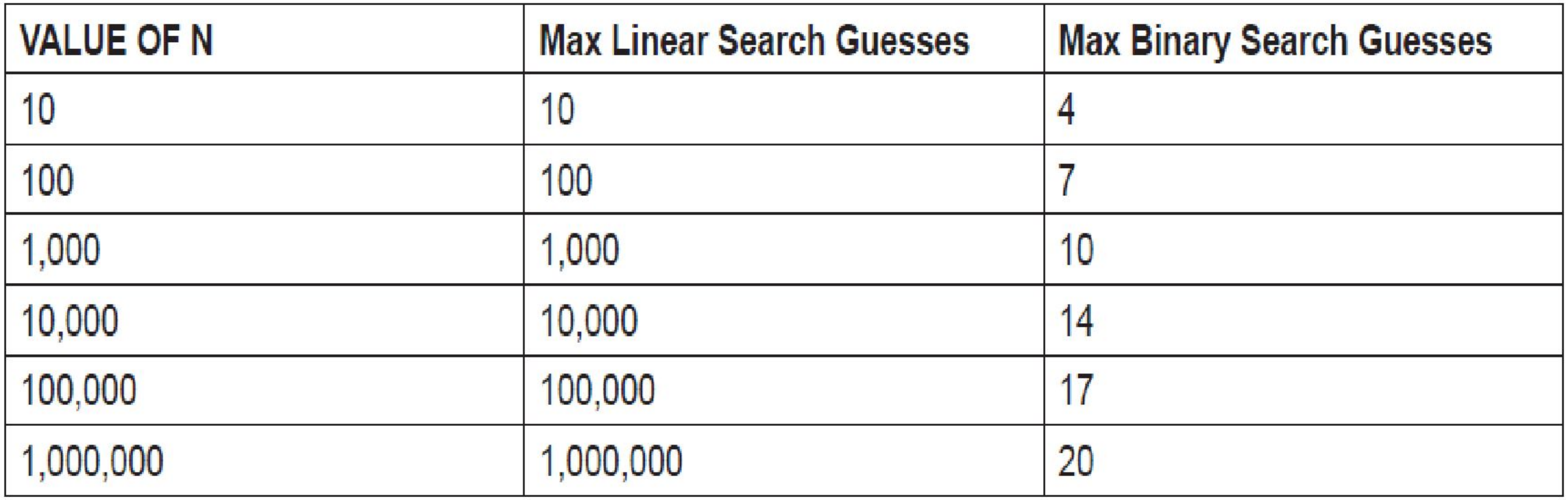
- Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands.
- Select the first unsorted element.
- Shift the other elements to the right to create the correct position and shift the unsorted elements to the correct position.
- Repeat until all the elements are sorted.

Insertion Sort Execution Example

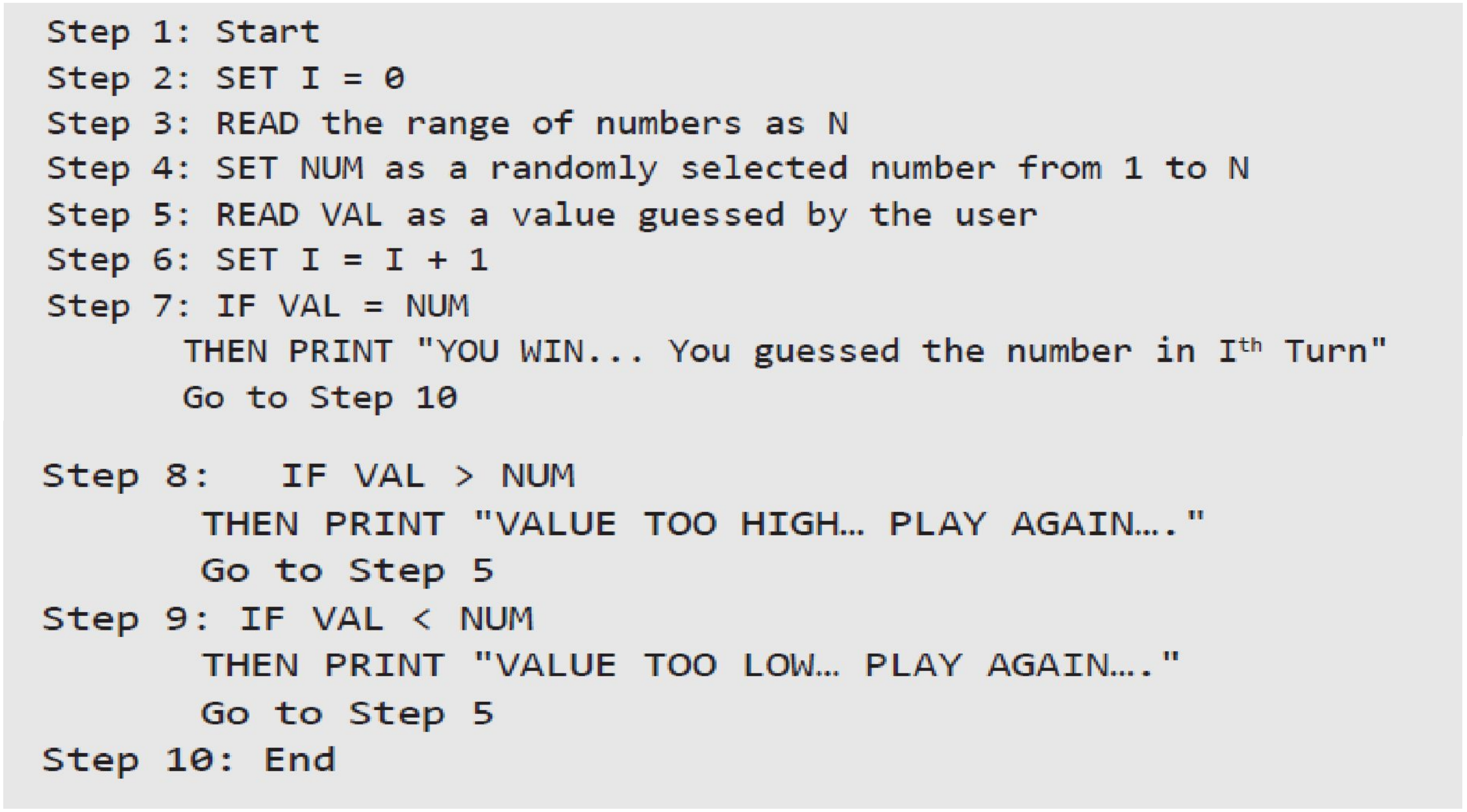


GUESS AN INTEGER NUMBER IN A RANGE

- Approaches
  - Linear search – Guess the number as 1,2,3,… guessed number.
    - Disadvantage: Compilation cost is high if the guessed number falls on the higher side.
  - Binary search – Guess by N/2 and the user tells the guessed value is > or <.
    - If < then eliminate N/2 to N
    - If > then eliminate 1 to N/2.

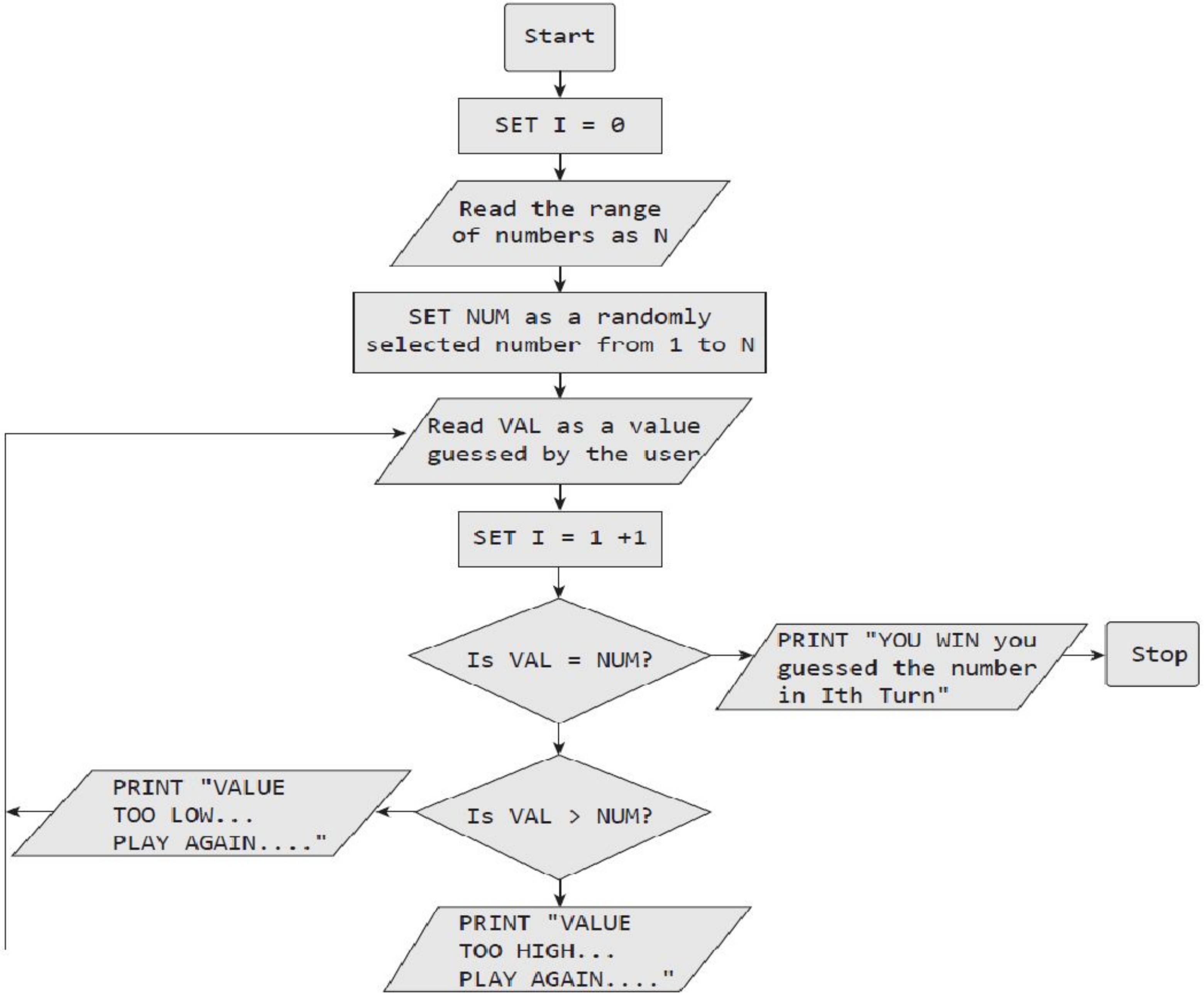


VALUE OF N	Max Linear Search Guesses	Max Binary Search Guesses
10	10	4
100	100	7
1,000	1,000	10
10,000	10,000	14
100,000	100,000	17
1,000,000	1,000,000	20



```
Step 1: Start
Step 2: SET I = 0
Step 3: READ the range of numbers as N
Step 4: SET NUM as a randomly selected number from 1 to N
Step 5: READ VAL as a value guessed by the user
Step 6: SET I = I + 1
Step 7: IF VAL = NUM
        THEN PRINT "YOU WIN... You guessed the number in Ith Turn"
        Go to Step 10

Step 8:   IF VAL > NUM
         THEN PRINT "VALUE TOO HIGH… PLAY AGAIN….."
         Go to Step 5
Step 9: IF VAL < NUM
         THEN PRINT "VALUE TOO LOW… PLAY AGAIN….."
         Go to Step 5
Step 10: End
```


Start
SET I = 0
Read the range of numbers as N
SET NUM as a randomly selected number from 1 to N
Read VAL as a value guessed by the user
SET I = 1 +1
Is VAL = NUM?
PRINT "YOU WIN you guessed the number in Ith Turn"
Stop
Is VAL > NUM?
PRINT "VALUE TOO LOW... PLAY AGAIN...."
PRINT "VALUE TOO HIGH... PLAY AGAIN...."

```
Start
Set I = 0
Read the range of numbers as N
Set NUM as a randomly selected number from 1 to N
Read VAL as a value guessed by the user
Set I = I + 1
While VAL != NUM
IF VAL = NUM
      THEN PRINT "YOU WIN... You guessed the number in $I^{th}$ Turn"
      Exit
ELSEIF VAL > NUM
      THEN PRINT "VALUE TOO HIGH… PLAY AGAIN….."
   ELSE
      THEN PRINT "VALUE TOO LOW… PLAY AGAIN….."
End
```