SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Artificial Intelligence and Data Science

Course Name – Computational Thinking and Python Programming

I Year / I Semester

Unit 1-Computational thinking and problem solving

Simple strategies for developing algorithm:

They are two commonly used strategies used in developing algorithm

- Iteration
- Recursion

Iteration

The iteration is when a loop repeatedly executes till the controlling condition becomes false

The iteration is applied to the set of instructions which we want to get repeatedly executed.

Iteration includes initialization, condition, and execution of statement within loop and update (increments and decrements) the control variable.

A sequence of statements is executed until a specified condition is true is called iterations.

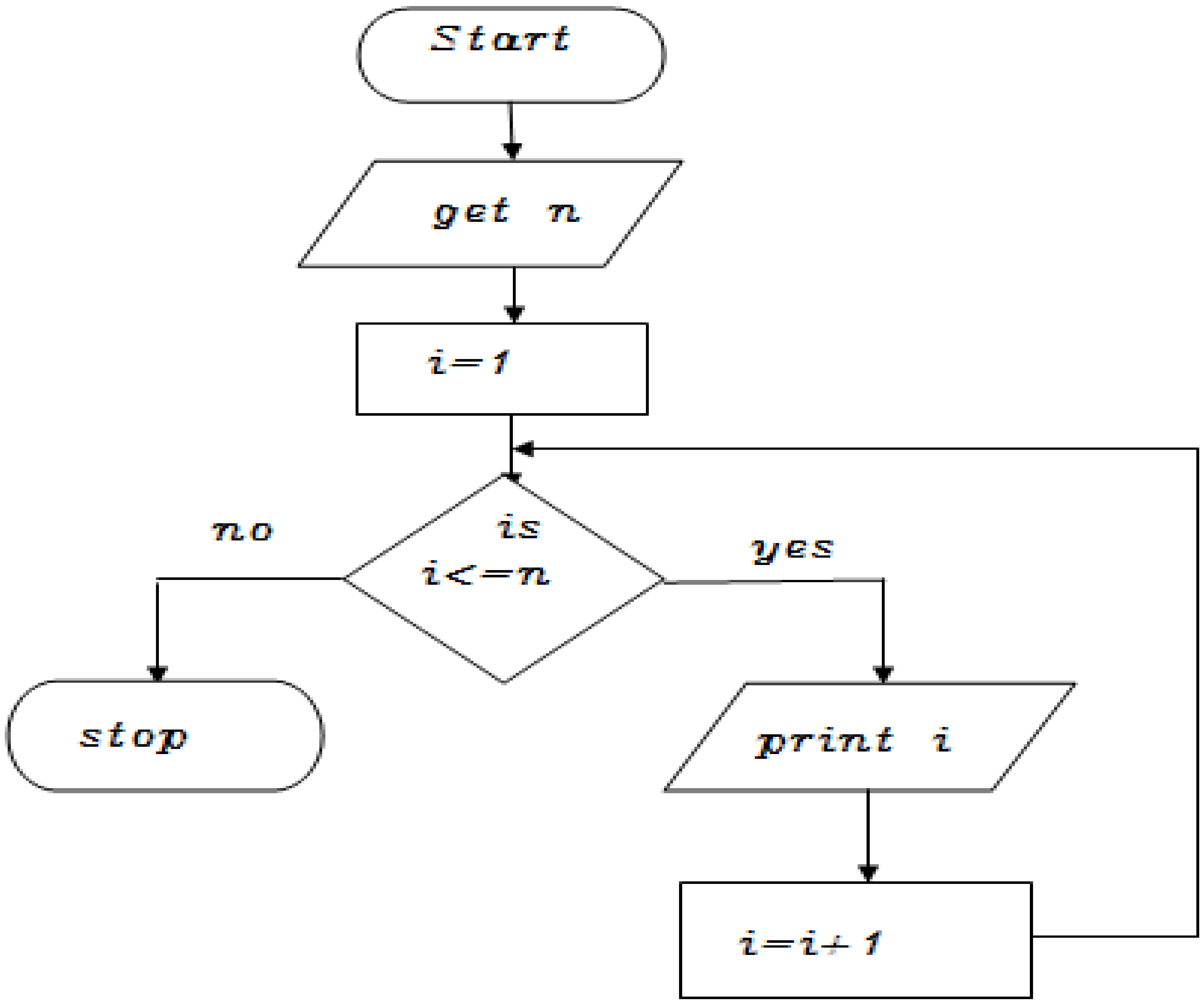
- for loop
- While loop

Syntax for For:	Example: Print n natural numbers

FOR(*start-value* to *end-value*) DO
statement
...ENDFOR
	BEGIN
	GET n
	INITIALIZE i=1
	FOR (i<=n) DO
	PRINT i
	i=i+
	1
	ENDFOR
	END

Syntax for While:	**Example: Print n natural numbers**

WHILE (condition) DO
statement
...
ENDWHILE
	BEGIN
	GET n
	INITIALIZE i=1
	WHILE(i<=n) DO
	PRINT i
	i=i+1
	ENDWHILE
	END


Start
get n
i=1
is
i<=n
no
yes
stop
print i
i=i+1

Recursions:

•A function that calls itself is known as recursion.

•Recursion is a process by which a function calls itself repeatedly until some specified condition has beensatisfied.

Algorithm for factorial of n numbers using recursion:

Main function:

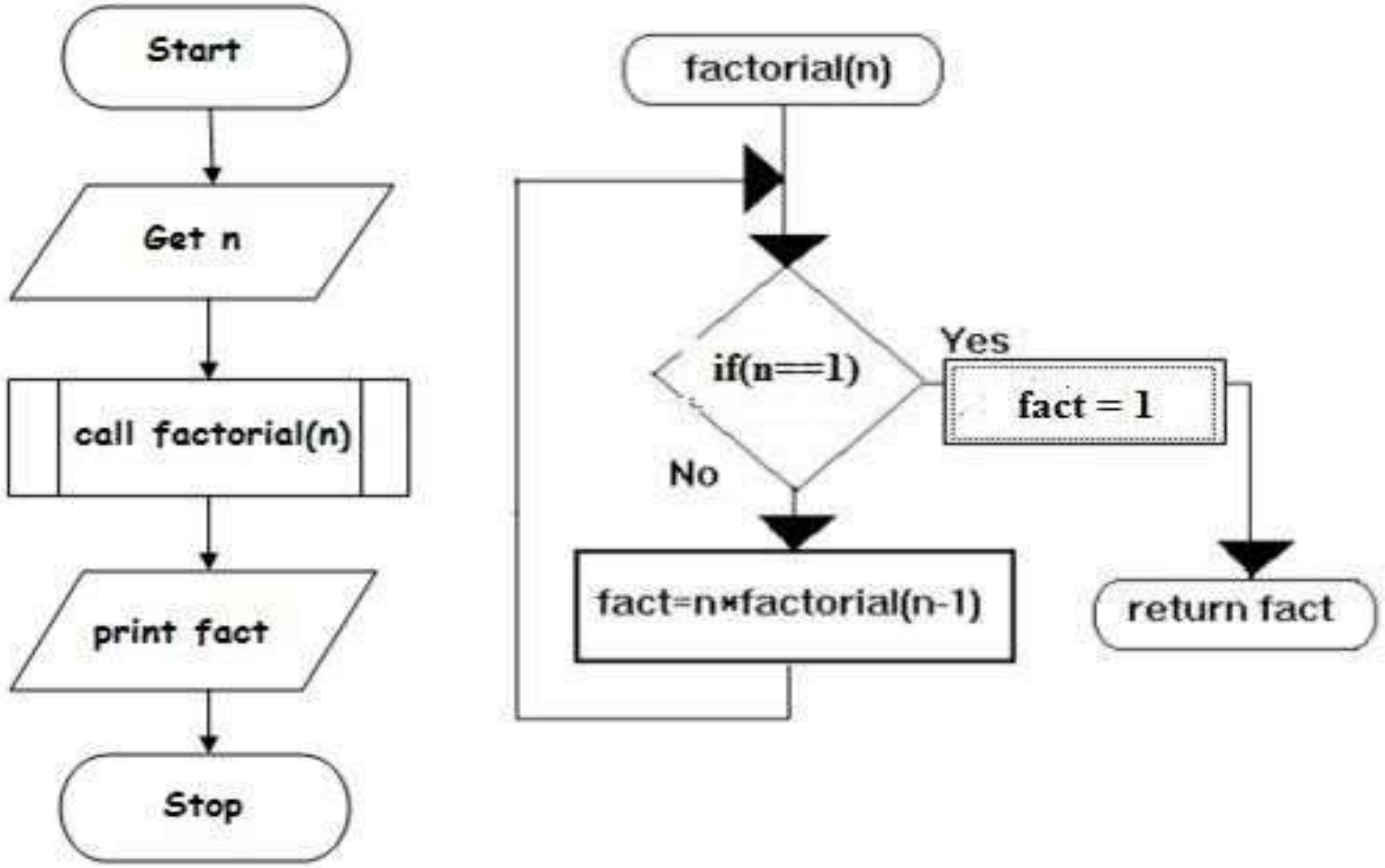
Step1: Start Step2: Get n

Step3: call factorial(n) Step4: print fact Step5: Stop

Sub function factorial(n):

Step1: if(n==1) then fact=1 return fact

Step2: else fact=n*factorial(n-1) and return fact



Pseudo code for factorial using recursion:

Main function:

```
BEGIN
GET n
CALL
factorial(n)
PRINT fact
BIN
```

Sub function factorial(n):

```
IF(n==1) THEN
        fact=1
        RETURN fact
ELSE
        RETURN fact=n*factorial(n-1)
```

<u>ILLUSTRATIVE PROBLEMS</u>

Guess an integer in a range

Algorithm:

Step1: Start

Step 2: Declare n, guess

Step 3: Compute guess=input Step 4: Read guess

Step 5: If guess>n, then

Print your guess is too high Else

Step6:If guess<n, then

Print your guess is too low Else

Step 7:If guess==n,then

Print Good job Else

Nope Step 6: Stop

Pseudocode:

BEGIN

COMPUTE guess=input READ guess,

IF guess>n

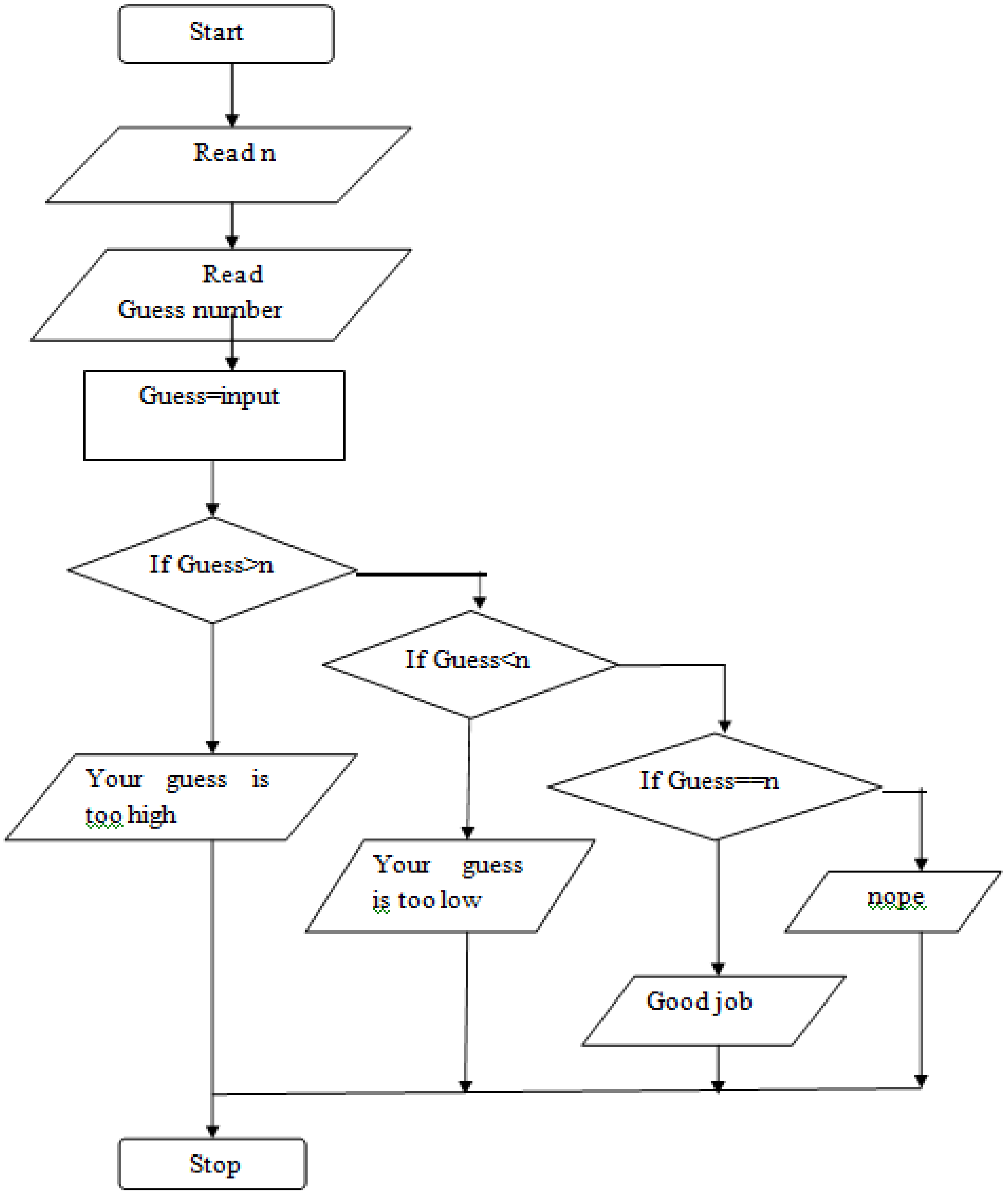
PRINT Guess is high ELSE

IF guess<n

PRINT Guess is low ELSE

IF guess=n PRINT Good job ELSE

Nope END

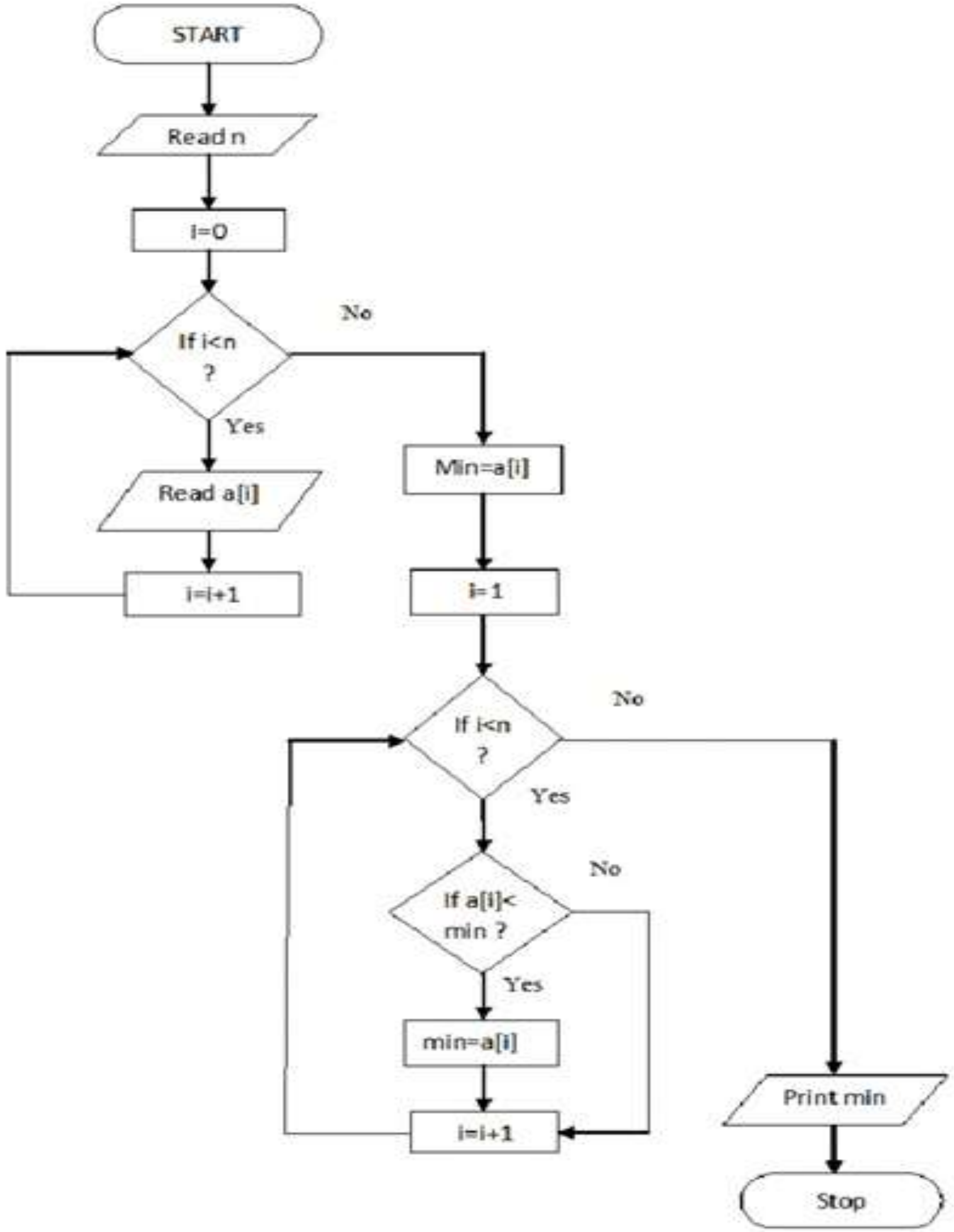

Start
Read n
Read
Guess number
Guess=input
If Guess>n
If Guess<n
If Guess==n
Your guess is too high
Your guess is too low
Good job
nope
Stop

Find minimum in a list Algorithm:

Step 1: Start Step 2: Read n
Step 3:Initialize i=0
Step 4: If i<n, then goto step 4.1, 4.2 else goto step 5
Step4.1: Read a[i]
Step 4.2: i=i+1 goto step 4
Step 5: Compute min=a[0]
Step 6: Initialize i=1
Step 7: If i<n, then go to step 8 else goto step 10
Step 8: If a[i]<min, then goto step 8.1,8.2 else goto 8.2
Step 8.1: min=a[i]
Step 8.2: i=i+1 goto 7
Step 9: Print min
Step 10: Stop

Pseudocode:

BEGIN
READ n
FOR i=0 to n, then
READ a[i]
INCREMENT i
END FOR
COMPUTE min=a[0]
FOR i=1 to n, then
IF a[i]<min, then
CALCULATE min=a[i]
INCREMENT i
ELSE INCREMENT i
END IF-ELSE
END FOR
PRINT min
END


START
Read n
i=0
If i<n ?
No
Yes
Read a[i]
i=i+1
Min=a[i]
i=1
If i<n ?
No
Yes
If a[i]< min ?
No
Yes
min=a[i]
i=i+1
Print min
Stop

PRINT min END

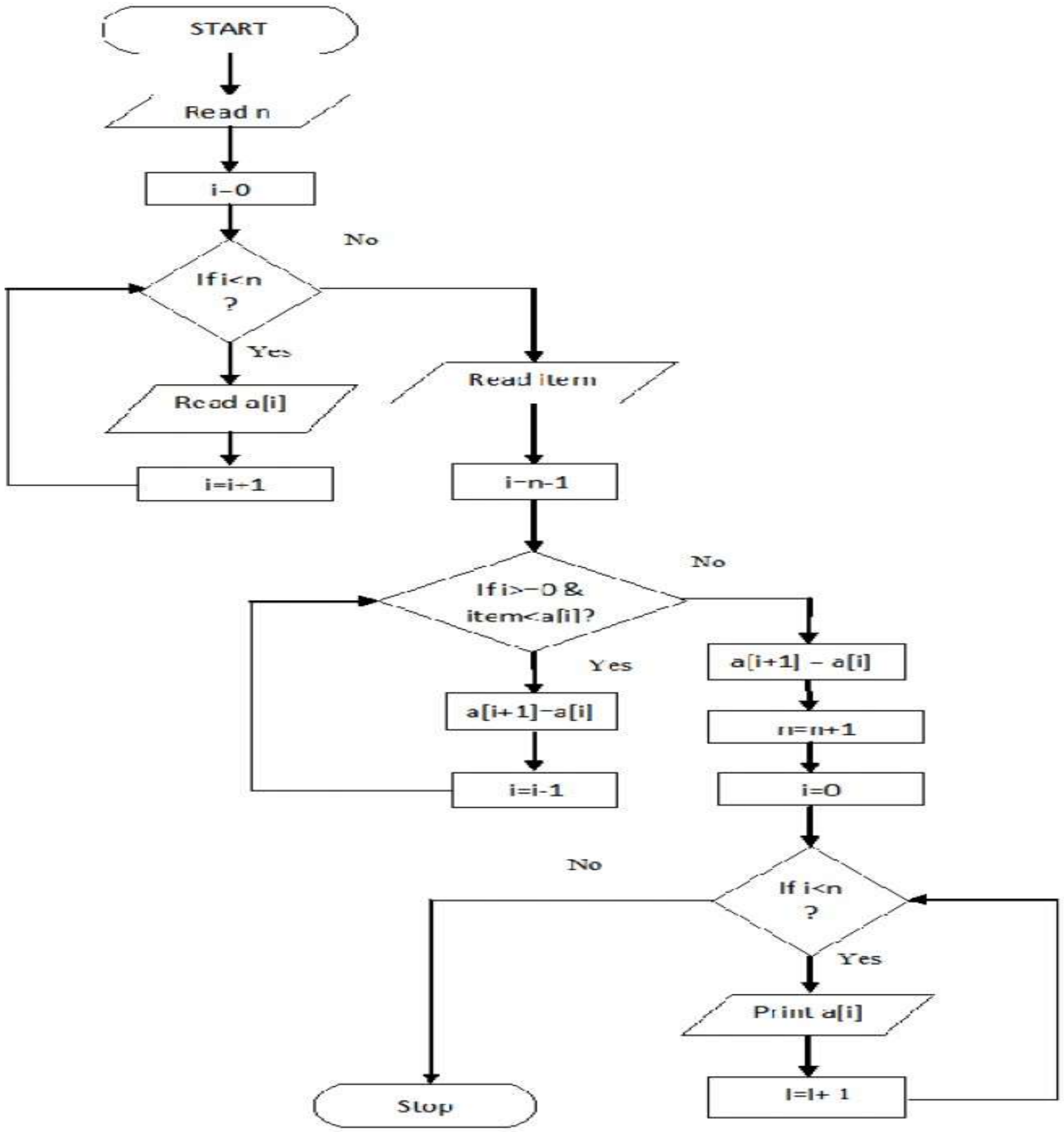
Insert a card in a list of sorted cards

Algorithm:

Step 1: Start Step 2: Read n
Step 3:Initialize i=0
Step 4: If i<n, then goto step 4.1, 4.2 else goto step 5
Step4.1: Read a[i]
Step 4.2: i=i+1 goto step 4
Step 5: Read item
Step 6: Calculate i=n-1
Step 7: If i>=0 and item<a[i], then go to step 7.1, 7.2 else goto step 8
Step 7.1: a[i+1]=a[i]
Step 7.2: i=i-1 goto step 7
Step 8: Compute a[i+1]=item
Step 9: Compute n=n+1
Step 10: If i<n, then goto step 10.1, 10.2 else goto step 11
Step10.1: Print a[i]
Step10.2: i=i+1 goto step 10
Step 11: Stop

Pseudocode:

BEGIN READ n
FOR i=0 to n, then
READ a[i]
INCREMENT i
END FOR
READ item
FOR i=n-1 to 0 and item<a[i], then
CALCULATE a[i+1]=a[i]
DECREMENT i
END FOR
COMPUTE a[i+1]=a[i]
COMPUTE n=n+1
FOR i=0 to n, then
PRINT a[i]
INCREMENT i
END FOR
END


START
Read n
i-0
No
If i<n ?
Yes
Read a[i]
i=i+1
Read item
i-n-1
No
If i>=0 & item<a[i]?
Yes
a[i+1]-a[i]
i=i-1
a[i+1] – a[i]
n=n+1
i=0
No
If i<n ?
Yes
Print a[i]
I=I+ 1
Stop

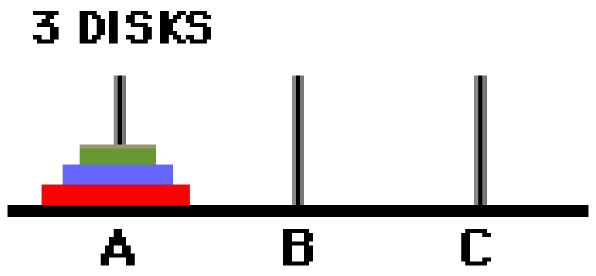
Tower of Hanoi

Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings.

Tower of Hanoi is one of the best example for recursive problem solving.

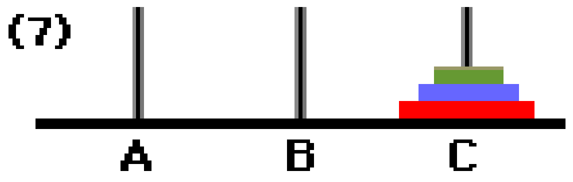
Pre-condition:

These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.



Post-condition:

All the disk should be moved to the last pole and placed only in ascending order as shown below.



Rules

The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are

Only one disk can be moved among the towers at any given time.

Only the "top" disk can be removed.

No large disk can sit over a small disk.

Tower of Hanoi puzzle with n disks can be solved in minimum $\mathbf{2^n - 1}$ steps.

This presentation shows that a puzzle with 3 disks has taken $\mathbf{2^3 - 1 = 7}$ steps.

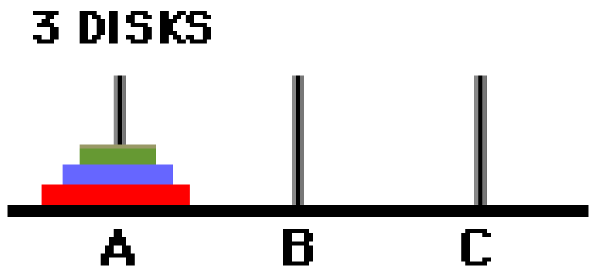
Tower of Hanoi

Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings.

Tower of Hanoi is one of the best example for recursive problem solving.

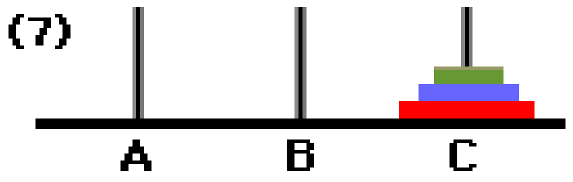
Pre-condition:

These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.



Post-condition:

All the disk should be moved to the last pole and placed only in ascending order as shown below.



Rules

The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are

Only one disk can be moved among the towers at any given time.

Only the "top" disk can be removed.

No large disk can sit over a small disk.

Tower of Hanoi puzzle with n disks can be solved in minimum $\mathbf{2^n-1}$ steps.

This presentation shows that a puzzle with 3 disks has taken $\mathbf{2^3 - 1 = 7}$ steps.

Algorithm

To write an algorithm for Tower of Hanoi, first we need to learn how to solve this problem with lesser amount of disks, say → 1 or 2. We mark three towers with name, **source**, **aux** (only to help moving the disks) and **destination**.

Input: one disk

If we have only one disk, then it can easily be moved from source to destination peg.

Input: two disks

If we have 2 disks −

First, we move the smaller (top) disk to aux peg.

Then, we move the larger (bottom) disk to destination peg.

And finally, we move the smaller disk from aux to destination peg.

Input: more than two disks

So now, we are in a position to design an algorithm for Tower of Hanoi with more than two disks. We divide the stack of disks in two parts. The largest disk (n^{th} disk) is in one part and all other (n-1) disks are in the second part.

Our ultimate aim is to move disk **n** from source to destination and then put all other (n1) disks onto it. We can imagine to apply the same in a recursive way for all given set of disks.

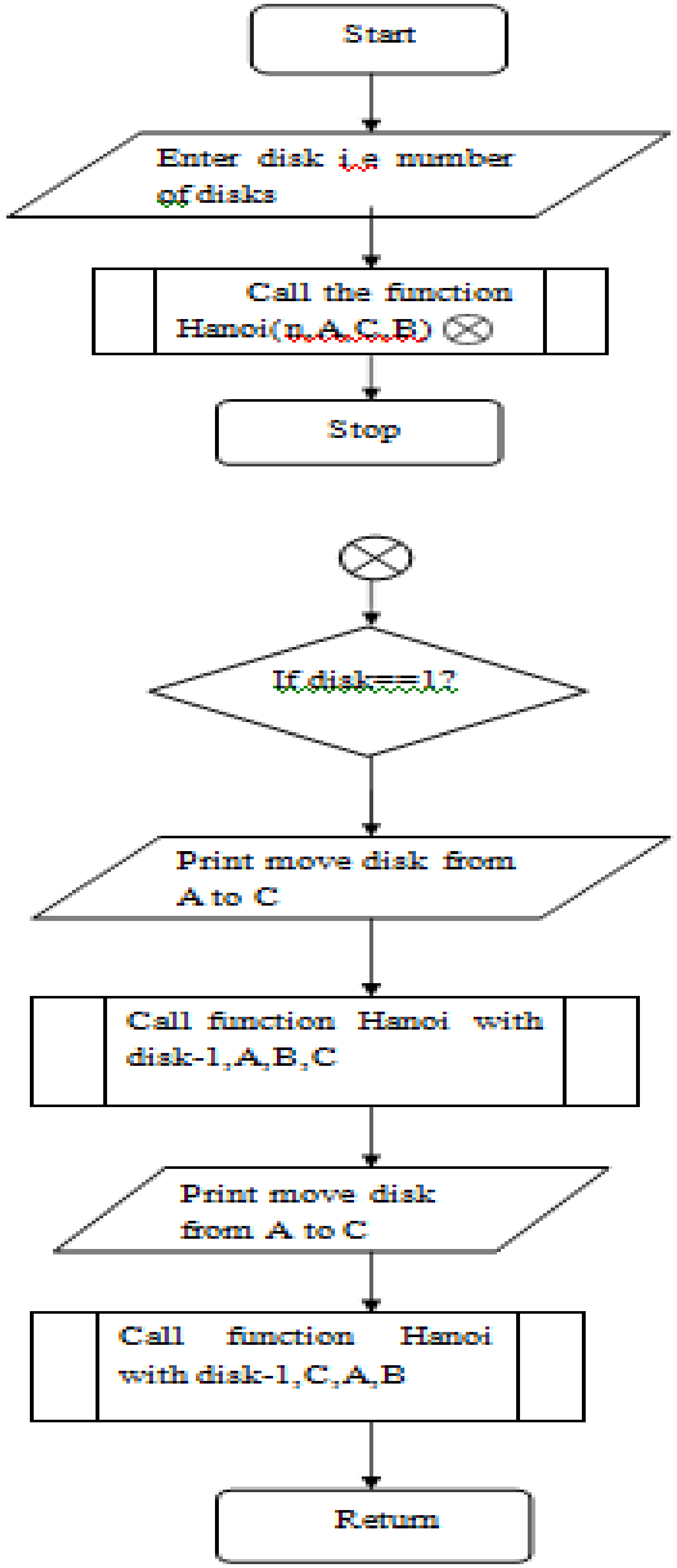
The steps to follow are –

Step 1 − Move n-1 disks from source to aux **Step 2** − Move nth disk from source to dest **Step 3** − Move n-1 disks from aux to dest

A recursive algorithm for Tower of Hanoi can be driven as follows –

```
START
Procedure Hanoi(disk, source, dest, aux)
IF disk == 1, THEN
          move disk from source to dest
ELSE
Hanoi(disk - 1, source, aux, dest) // Step 1
move disk from source to dest // Step 2
Hanoi(disk - 1, aux, dest, source) // Step 3
END IF
END Procedure STOP
```

FLOW CHART


Start
Enter disk i.e number of disks
Call the function Hanoi(n,A,C,B)
Stop
If disk==1?
Print move disk from A to C
Call function Hanoi with disk-1,A,B,C
Print move disk from A to C
Call function Hanoi with disk-1,C,A,B
Return