GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING

SYLLABUS

OBJECTIVES:

- To know the basics of algorithmic problem solving
- To read and write simple Python programs.
- To develop Python programs with conditionals and loops.
- To define Python functions and call them.
- To use Python data structures –- lists, tuples, dictionaries.
- To do input/output with files in Python.

UNIT I ALGORITHMIC PROBLEM SOLVING 9

Algorithms, building blocks of algorithms (statements, state, control flow, functions), notation (pseudo code, flow chart, programming language), algorithmic problem solving, simple strategies for developing algorithms (iteration, recursion). Illustrative problems: find minimum in a list, insert a card in a list of sorted cards, guess an integer number in a range, Towers of Hanoi.

UNIT II DATA, EXPRESSIONS, STATEMENTS 9

Python interpreter and interactive mode; values and types: int, float, boolean, string, and list; variables, expressions, statements, tuple assignment, precedence of operators, comments; modules and functions, function definition and use, flow of execution, parameters and arguments; Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points.

UNIT III CONTROL FLOW, FUNCTIONS 9

Conditionals: Boolean values and operators, conditional (if), alternative (if-else), chained conditional (if-elif-else); Iteration: state, while, for, break, continue, pass; Fruitful functions: return values, parameters, local and global scope, function composition, recursion; Strings: string slices, immutability, string functions and methods, string module; Lists as arrays. Illustrative programs: square root, gcd, exponentiation, sum an array of numbers, linear search, binary search.

UNIT IV LISTS, TUPLES, DICTIONARIES 9

Lists: list operations, list slices, list methods, list loop, mutability, aliasing, cloning lists, list parameters; Tuples: tuple assignment, tuple as return value; Dictionaries: operations and methods; advanced list processing - list comprehension; Illustrative programs: selection sort, insertion sort, mergesort, histogram.

UNIT V FILES, MODULES, PACKAGES 9

Files and exception: text files, reading and writing files, format operator; command line arguments, errors and exceptions, handling exceptions, modules, packages; Illustrative programs: word count, copy file.

TOTAL : 45 PERIODS

TEXT BOOKS

Allen B. Downey, ‘‘Think Python: How to Think Like a Computer Scientist‘‘, 2nd edition, Updatedfor Python 3,Shroff/O‘Reilly Publishers, 2016 (http://greenteapress.com/wp/thinkpython/)

Guido van Rossum and Fred L. Drake Jr, —An Introduction to Python – Revised and updated for Python 3.2, Network Theory Ltd., 2011.

REFERENCES:

John V Guttag, —Introduction to Computation and Programming Using Python‘‘, Revised and expanded Edition, MIT Press , 2013

Robert Sedgewick, Kevin Wayne, Robert Dondero, —Introduction to Programming in Python: An Inter-disciplinary Approach, Pearson India Education Services Pvt. Ltd., 2016.

UNIT I ALGORITHMIC PROBLEM SOLVING

INTRODUCTION

PROBLEM SOLVING

Problem solving is the systematic approach to define the problem and creating number of solutions.

The problem solving process starts with the problem specifications and ends with a correct program.

PROBLEM SOLVING TECHNIQUES

Problem solving technique is a set of techniques that helps in providing logic for solving a problem.

Problem solving can be expressed in the form of

1. Algorithms.
2. Flowcharts.
3. Pseudo codes.
4. Programs

1.ALGORITHM

It is defined as a sequence of instructions that describe a method for solving a problem. In other words it is a step by step procedure for solving a problem

- Should be written in simple English
- Each and every instruction should be precise and unambiguous.
- Instructions in an algorithm should not be repeated infinitely.
- Algorithm should conclude after a finite number of steps.
- Should have an end point
- Derived results should be obtained only after the algorithm terminates.

Qualities of a good algorithm

The following are the primary factors that are often used to judge the quality of the algorithms.

Time – To execute a program, the computer system takes some amount of time. The lesser is the time required, the better is the algorithm.

Memory – To execute a program, computer system takes some amount of memory space. The lesser is the memory required, the better is the algorithm.

Accuracy – Multiple algorithms may provide suitable or correct solutions to a given problem, some of these may provide more accurate results than others, and such algorithms may be suitable

Building Blocks of Algorithm

As algorithm is a part of the blue-print or plan for the computer program. An algorithm is constructed using following blocks.

- Statements
- States
- Control flow
- Function

Statements

Statements are simple sentences written in algorithm for specific purpose. Statements may consists of **assignment statements, input/output statements, comment statements**

Example:

- Read the value of 'a' //This is input statement
- Calculate c=a+b //This is assignment statement
- Print the value of c // This is output statement

Comment statements are given after // symbol, which is used to tell the purpose of the line.

States

An algorithm is deterministic automation for accomplishing a goal which, given an initial state, will terminate in a defined end-state.

An algorithm will **definitely** have **start state** and **end state**.

Control Flow

Control flow which is also stated as flow of control, determines what section of code is to run in program at a given time. There are three types of flows, they are

1. Sequential control flow
2. Selection or Conditional control flow
3. Looping or repetition control flow

Sequential control flow:

The name suggests the sequential control structure is used to perform the action one after another. Only one step is executed once. The logic is top to bottom approach.

Example

Description: To find the sum of two numbers.

1. Start
2. Read the value of 'a'
3. Read the value of 'b'
4. Calculate sum=a+b
5. Print the sum of two number
6. Stop

Selection or Conditional control flow

Selection flow allows the program to make choice between two alternate paths based on condition. It is also called as **decision structure**

Basic structure:

```
IFCONDITION is TRUE then
    perform some action
ELSE IF CONDITION is FALSE then
    perform some action
```

The conditional control flow is explained with the example of finding greatest of two numbers.

Example

Description: finding the greater number

1. Start
2. Read a

```
3. Read b
4. If a>b then
    4.1. Print a is greater
else
    4.2. Print b is greater
5. Stop
```

Repetition control flow

Repetition control flow means that one or more steps are performed repeatedly until some condition is reached. This logic is used for producing loops in program logic when one one more instructions may need to be executed several times or depending on condition.

Basic Structure:

```
Repeat until CONDITION is true
    Statements
```

Example

Description: to print the values from 1 to n

```
1. Start
2. Read the value of ‘n’
3. Initialize i as 1
4. Repeat step 4.1 until i< n

    4.1. Print i
5. Stop
```

Function

A function is a block of organized, reusable code that is used to perform a single, related action. Function is also named as **methods, sub-routines**.

Elements of functions:

1. Name for declaration of function
2. Body consisting local declaration and statements
3. Formal parameter
4. Optional result type.

Basic Syntax

```
function_name(parameters)
    function statements
end function
```

<u>Algorithm for addition of two numbers using function</u>

<u>Main function()</u>

```
Step 1: Start
Step 2: Call the function add()
Step 3: Stop
```

<u>sub function add()</u>

```
Step 1: Function start
```

Step2: Geta,bValues
Step 3: add c=a+b
Step 4: Printc
Step 5: Return

2.NOTATIONS OF AN ALGORITHM

Algorithm can be expressed in many different notations, including **Natural Language, Pseudo code, flowcharts and programming languages**. Natural language tends to be verbose and ambiguous. Pseudocode and flowcharts are represented through structured human language.

A notation is a system of characters, expressions, graphics or symbols designs used among each others in problem solving to represent technical facts, created to facilitate the best result for a program

Pseudocode

Pseudocode is an **informal high-level description** of the operating principle of a **computer program** or **algorithm**. It uses the basic structure of a normal programming language, but is intended for human reading rather than machine reading.

It is text based detail design tool. **Pseudo means false** and **code** refers to **instructions written in programming language**.

Pseudocode cannot be compiled nor executed, and there are no real formatting or syntax rules. The pseudocode is written in normal English language which cannot be understood by the computer.

Example:

Pseudocode: To find sum of two numbers

```
READ num1,num2
sum=num1+num2
PRINT sum
```

Basic rules to write pseudocode:

1. Only one statement per line.
   Statements represents single action is written on same line. For example to read the input, all the inputs must be read using single statement.
2. Capitalized initial keywords
   The keywords should be written in capital letters. Eg: **READ, WRITE, IF, ELSE, ENDIF, WHILE, REPEAT, UNTIL**
   Example:
   Pseudocode: Find the total and average of three subjects
   ```
   RAED name, department, mark1, mark2, mark3
   Total=mark1+mark2+mark3
   Average=Total/3
   WRITE name, department,mark1, mark2, mark3
   ```
3. Indent to show hierarchy
   Indentation is a process of showing the boundaries of the structure.
4. End multi-line structures
   Each structure must be ended properly, which provides more clarity.
   Example:

```
Pseudocode: Find greatest of two numbers
READ a, b
IF a>b then
    PRINT a is greater
ELSE
    PRINT b is greater
ENDIF
```

5. Keep statements language independent.
   Pesudocode must never written or use any syntax of any programming language.

Advantages of Pseudocode

- Can be done easily on a word processor
- Easily modified
- Implements structured concepts well
- It can be written easily
- It can be read and understood easily
- Converting pseudocode to programming language is easy as compared with flowchart

Disadvantages of Pseudocode

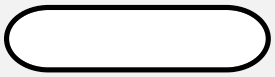
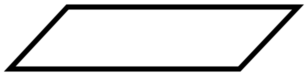
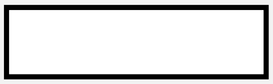
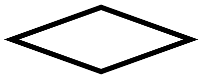
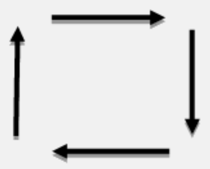
- It is not visual
- There is no standardized style or format

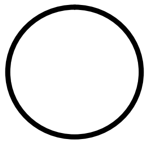
Flowchart

A **graphical representation** of an algorithm. Flowcharts is a diagram made up of boxes, diamonds, and other shapes, connected by arrows.

Each shape represents a step in process and arrows show the order in which they occur.

Table 1: Flowchart Symbols

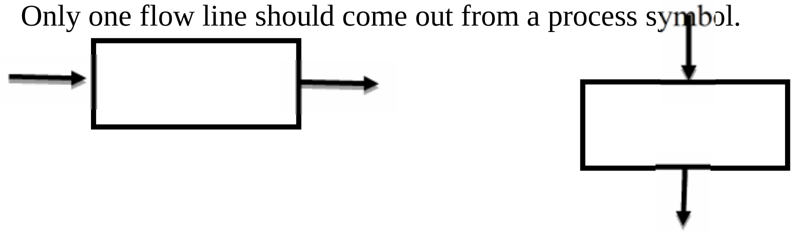
S.No	Name of symbol	Symbol	Type	Description
1.	Terminal Symbol		Oval	Represent the start and stop of the program.
2.	Input/ Output symbol		Parallelogram	Denotes either input or output operation.
3.	Process symbol		Rectangle	Denotes the process to be carried
4.	Decision symbol		Diamond	Represents decision making and branching
5.	Flow lines		Arrow lines	Represents the sequence of steps and direction of flow. **Used to connect symbols**.

6.	Connector		Circle	A connector symbol is represented by a circle and a letter or digit is placed in the circle to specify the link. This symbol is **used to connect flowcharts**.

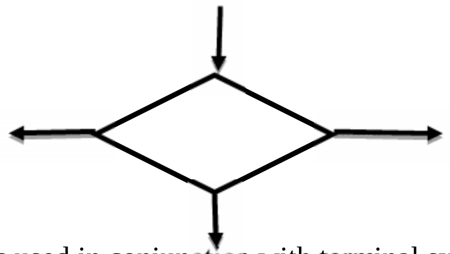
Rules for drawing flowchart

1. In drawing a proper flowchart, all necessary requirements should be listed out in logical order.
2. The flow chart should be clear, neat and easy to follow. There should not be any room for ambiguity in understanding the flowchart.
3. The usual directions of the flow of a procedure or system is from left to right or top to bottom.

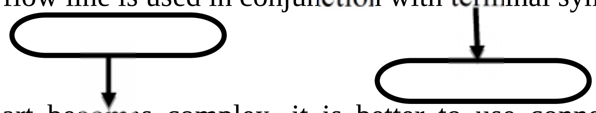
   Only one flow line should come out from a process symbol.



4. Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, cap leave the decision symbol.



5. Only one flow line is used in conjunction with terminal symbol.



6. If flowchart becomes complex, it is better to use connector symbols to reduce the number of flow lines.
7. Ensure that flowchart has logical start and stop.

<u>**Advantages of Flowchart**</u>

Communication:

Flowcharts are better way of communicating the logic of the system.

Effective Analysis

With the help of flowchart, a problem can be analyzed in more effective way.

Proper Documentation

Flowcharts are used for good program documentation, which is needed for various purposes.

Efficient Coding

The flowcharts act as a guide or blue print during the system analysis and program development phase.

Systematic Testing and Debugging

The flowchart helps in testing and debugging the program

Efficient Program Maintenance

The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

Disadvantages of Flowchart

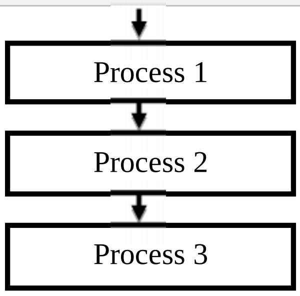
Complex Logic: Sometimes, the program logic is quite complicated. In that case flowchart becomes complex and difficult to use.

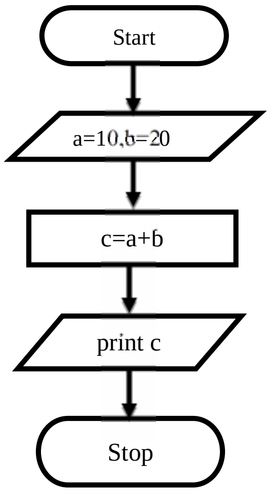
Alteration and Modification: If alterations are required the flowchart may require re-drawing completely.

Reproduction: As the flowchart symbols cannot be typed, reproduction becomes problematic.

Control Structures using flowcharts and Pseudocode

Sequence Structure

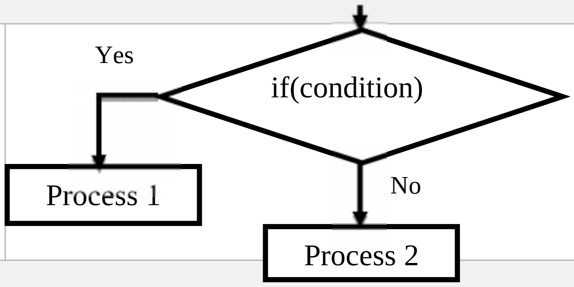
Pseudocode	Flow Chart
General Structure	
Process 1 …. Process 2 … Process 3	
Example	

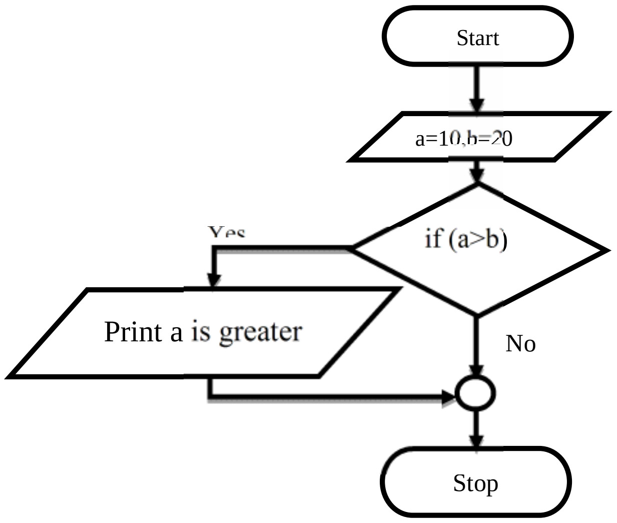
READ a READ b Result c=a+b PRINT c	

Conditional Structure

- Conditional structure is used to check the condition. It will be having two outputs only (True or False)
- **IF** and **IF…ELSE** are the conditional structures used in Python language.
- **CASE** is the structure used to select multi way selection control. It is not supported in Python.

Pseudocode	**Flow Chart**
General Structure	

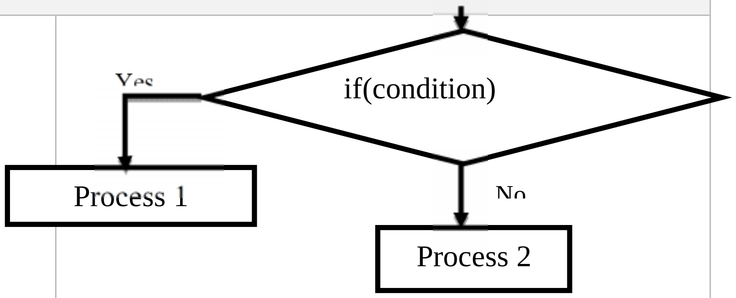
IF condition THEN
        Process 1
ENDIF
	

Example	
READ a READ b IF a>b THEN PRINT a is greater	 Start a=10,b=20 Yes if (a>b) Print a is greater No Stop

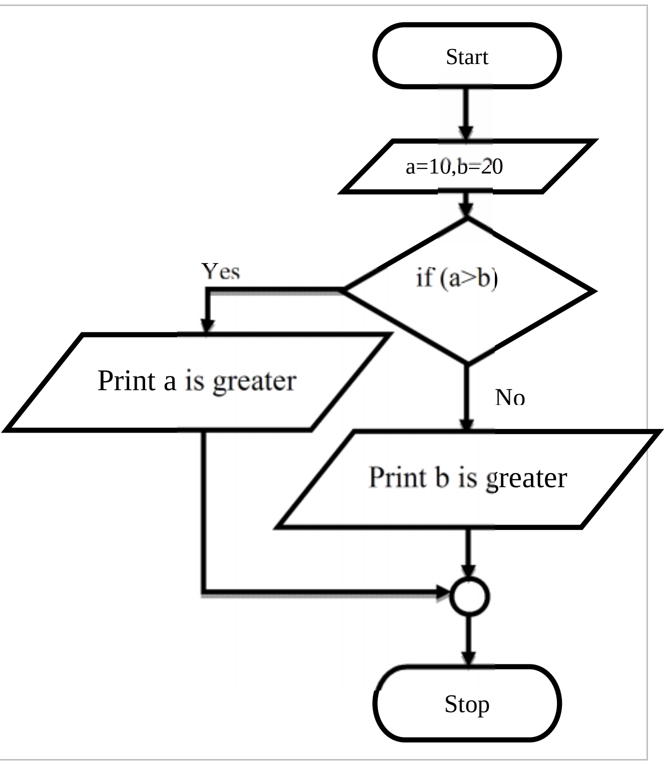
IF… ELSE

IF…THEN…ELSE is the structure used to specify, if the condition is true, then execute Process1, else, that is condition is false then execute Process2

Pseudocode	**Flow Chart**
General Structure	

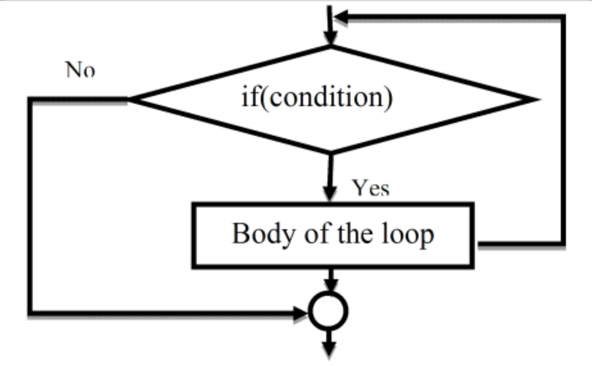
IF condition THEN
        Process 1
ELSE
        Process 2
ENDIF
	

Example	

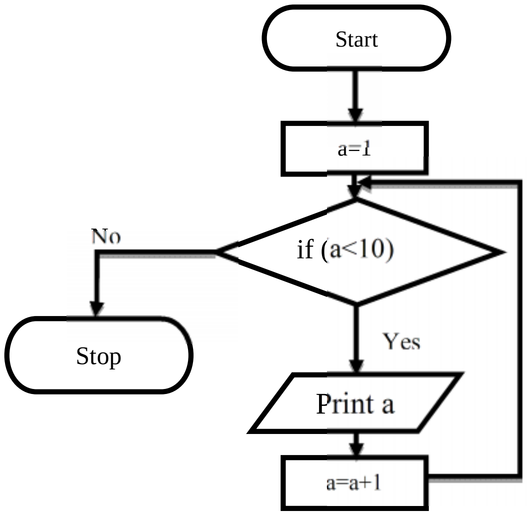
READ a READ b IF a>b THEN PRINT a is greater	 Start a=10,b=20 if (a>b) Yes Print a is greater No Print b is greater Stop

Iteration or Looping Structure

- Looping is generally used with **WHILE** or **DO...WHILE** or **FOR** loop.
- WHILE and FOR is entry checked loop

Pseudocode	**Flow Chart**
General Structure	
WHILE condition Body of the loop ENDWHILE	
Example	

- DO…WHILE is exit checked loop, so the loop will be executed at least once.

INITIALIZE a=1
WHILE a<10 THEN
    PRINT a
    a=a+1
ENDWHILE
	 Start
	a=1
	if (a<10)
	No
	Stop
	Yes
	Print a
	a=a+1

- In python DO…WHILE is not supported.
- If the loop condition is true then the loop gets into infinite loop, which may lead to system crash

Programming Language

- A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks. In other word it is set of instructions for the computer to solve the problem.
- Programming Language is a formal language with set of instruction, to the computer to solve a problem. The program will accept the data to perform computation.

Program= Algorithm +Data

Need for Programming Languages

- Programming languages are also used to organize the computation
- Using Programming language we can solve different problems
- To improve the efficiency of the programs.

Types of Programming Language

In general Programming languages are classified into three types. They are

- Low – level or Machine Language
- Intermediate or Assembly Language
- High – level Programming language

Machine Language:

Machine language is the lowest-level programming language (except for computers that utilize programmable microcode). Machine languages are the only languages understood by computers. It is also called as low level language.

Example code:100110011
111001100

Assembly Language:

An assembly language contains the same instructions as a machine language, but the instructions and variables have names instead of being just numbers. An assembler language consists of mnemonics, mnemonics that corresponds unique machine instruction.

Example code: start
addx,y
subx,y

High – level Language:

A high-level language (HLL) is a programming language such as C, FORTRAN, or Pascal that enables a programmer to write programs that are more or less independent of a particular type of computer. Such languages are considered high-level because they are closer to human languages and further from machine languages. Ultimately, programs written in a high-level language must be translated into machine language by a compiler or interpreter.

Example code: print(“Hello World!”)

High level programming languages are further divided as mentioned below.

Language Type	**Example**
Interpreted Programming Language	Python, BASIC, Lisp
Functional Programming Language	Clean, Curry, F#
Compiled Programming Language	C++,Java, Ada, ALGOL
Procedural Programming Language	C,Matlab, CList
Scripting Programming Language	PHP,Apple Script, Javascript
Markup Programming Language	HTML,SGML,XML
Logical Programming Language	Prolog, Fril
Concurrent Programming Language	ABCL, Concurrent PASCAL
Object Oriented Programming Language	C++,Ada, Java, Python

Interpreted Programming Language:

Interpreter is a program that executes instructions written in a high-level language.

An interpreter reads the source code one instruction or one line at a time, converts this line into machine code and executes it.

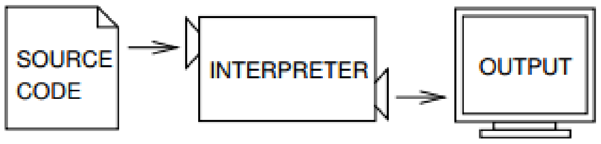


Figure : Interpreter

Compiled Programming Languages

Compile is to transform a program written in a high-level programming language from source code into object code. This can be done by using a tool called compiler.

A compiler reads the whole source code and translates it into a complete machine code program to perform the required tasks which is output as a new file.

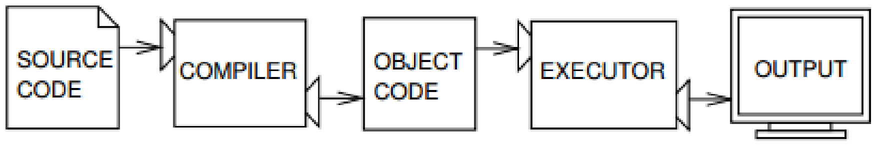


Figure: Compiler

Interpreted vs. Compiled Programming Language

Interpreted Programming Language	Compile Programming Language
Translates one statement at a time	Scans entire program and translates it as whole into machine code
It takes less amount of time to analyze the source code but the overall execution time is slower	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster
No intermediate object code is generated, hence are memory efficient	Generates intermediate object code which further requires linking, hence requires more memory
Continues translating the program until first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Eg: Python, Ruby	Eg: C,C++,Java

3.ALGORITHMIC PROBLEM SOLVING:

Algorithmic problem solving is solving problem that require the formulation of an algorithm for the solution.

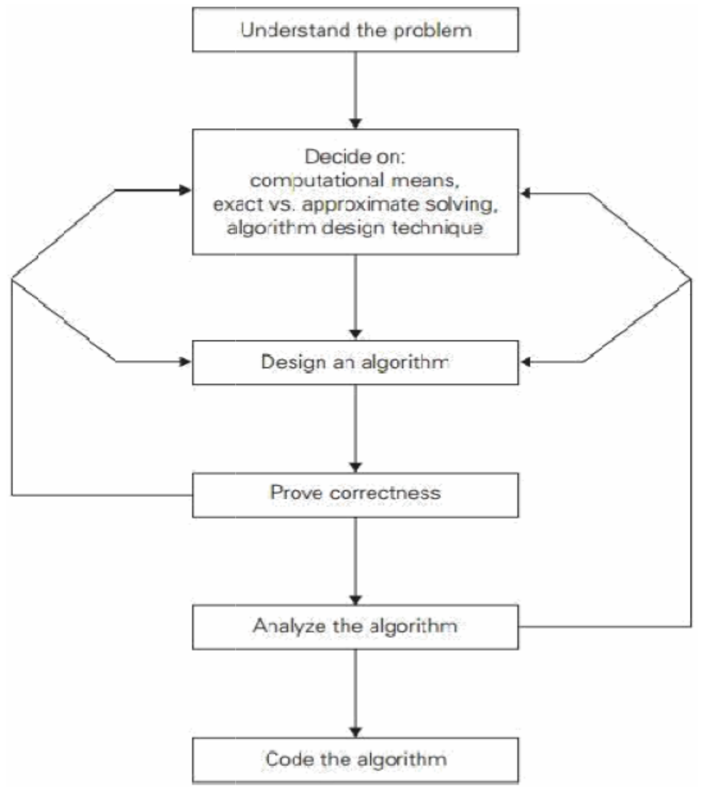


FIGURE 1.2 Algorithm design and analysis process.

Understanding the Problem

- It is the process of finding the input of the problem that the algorithm solves.
- It is very important to specify exactly the set of inputs the algorithm needs to handle.
- A correct algorithm is not one that works most of the time, but one that works Correctly for *all* legitimate inputs.

Ascertaining the Capabilities of the Computational Device

If the instructions are executed one after another, it is called sequential algorithm

Choosing between Exact and Approximate Problem Solving

- The next principal decision is to choose between solving the problem exactly or solving it approximately.
- Based on this, the algorithms are classified as exact *algorithm* and *approximation algorithm.*
- Data structure plays a vital role in designing and analysis the algorithms.
- Some of the algorithm design techniques also depend on the structuring data specifying a problem's instance
- Algorithm+ Data structure=programs.

Algorithm Design Techniques

- An ***algorithm design technique*** (or "strategy" or "paradigm") is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
- Learning these techniques is of utmost importance for the following reasons.
- First, they provide guidance for designing algorithms for new problems,
  Second, algorithms are the cornerstone of computer science.

Methods of Specifying an Algorithm

- ***Pseudocode*** is a mixture of a natural language and programming language-like constructs. Pseudocode is usually more precise than natural language, and its usage often yields more succinct algorithm descriptions.In the earlier days of computing, the dominant vehicle for specifying algorithms was a ***flowchart***, a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.

- **Programming language** can be fed into an electronic computer directly. Instead, it needs to be converted into a computer program written in a particular computer language. We can look at such a program as yet another way of specifying the algorithm, although it is preferable to consider it as the algorithm's implementation.

- Once an algorithm has been specified, you have to prove its ***correctness***. That is, you have to prove that the algorithm yields a required result for every legitimate input in a finite amount of time.

- A common technique for proving correctness is to use mathematical induction because an algorithm's iterations provide a natural sequence of steps needed for such proofs.

- It might be worth mentioning that although tracing the algorithm's performance for a few specific inputs can be a very worthwhile activity, it cannot prove the algorithm's correctness conclusively. But in order to show that an algorithm is incorrect, you need just one instance of its input for which the algorithm fails.

Analyzing an Algorithm

1. ***Efficiency***.
   Time efficiency: indicating how fast the algorithm runs,
   Space efficiency: indicating how much extra memory it uses

2. ***simplicity***.
   - An algorithm should be precisely defined and investigated with mathematical expressions.
   - Simpler algorithms are easier to understand and easier to program.
   - Simple algorithms usually contain fewer bugs.

Coding an Algorithm

- Most algorithms are destined to be ultimately implemented as computer programs. Programming an algorithm presents both a peril and an opportunity.
- A working program provides an additional opportunity in allowing an empirical analysis of the underlying algorithm. Such an analysis is based on timing the program on several inputs and then analyzing the results obtained.

4.Simple strategies for developing algorithm:

They are two commonly used strategies used in developing algorithm

1. Iteration
2. 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.

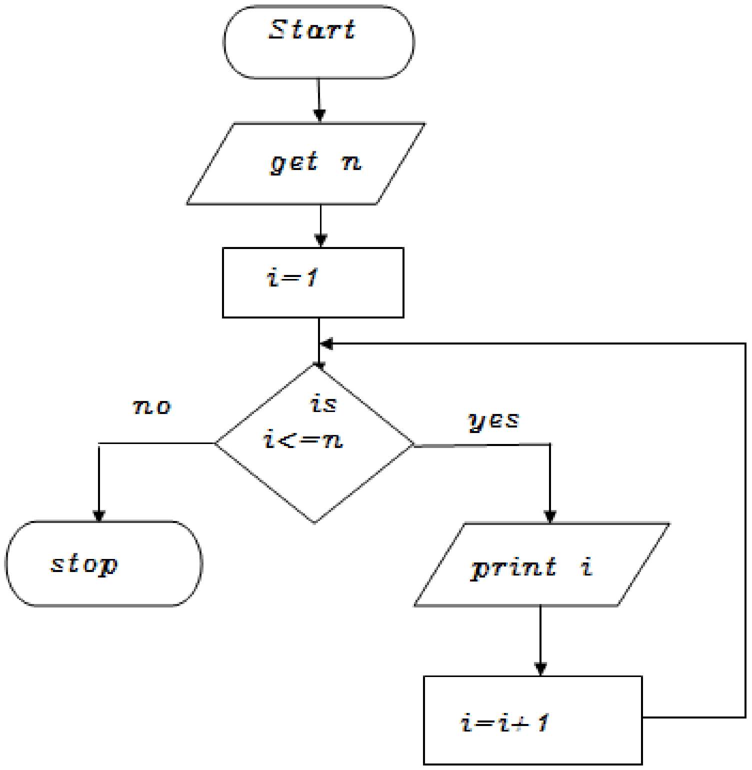
1. for loop
2. While loop

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



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 been satisfied.

Algorithm for factorial of n numbers using recursion:

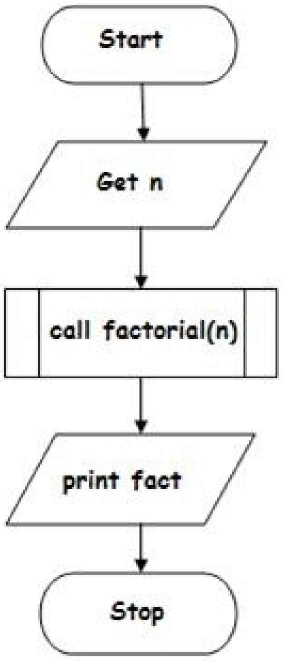
Main function:

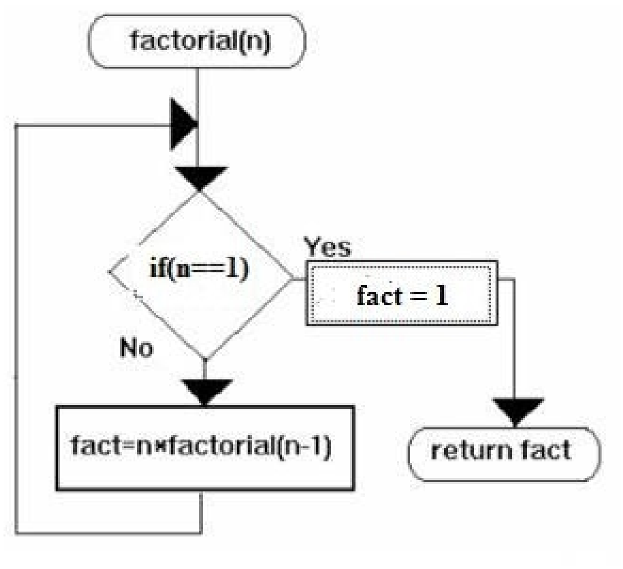
```
Step1: Start
Step2: Get n
Step3: call factorial(n)
Step4: print fact
Step5: Stop
```

Sub function factorial(n):

```
Step1: if(n==1) then fact=1 return fact
Step2: else fact=n*factorial(n-1) and return fact
```

FLOW CHART





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)
```

5. ILLUSTRATIVE PROBLEMS

1. Guess an integer in a range

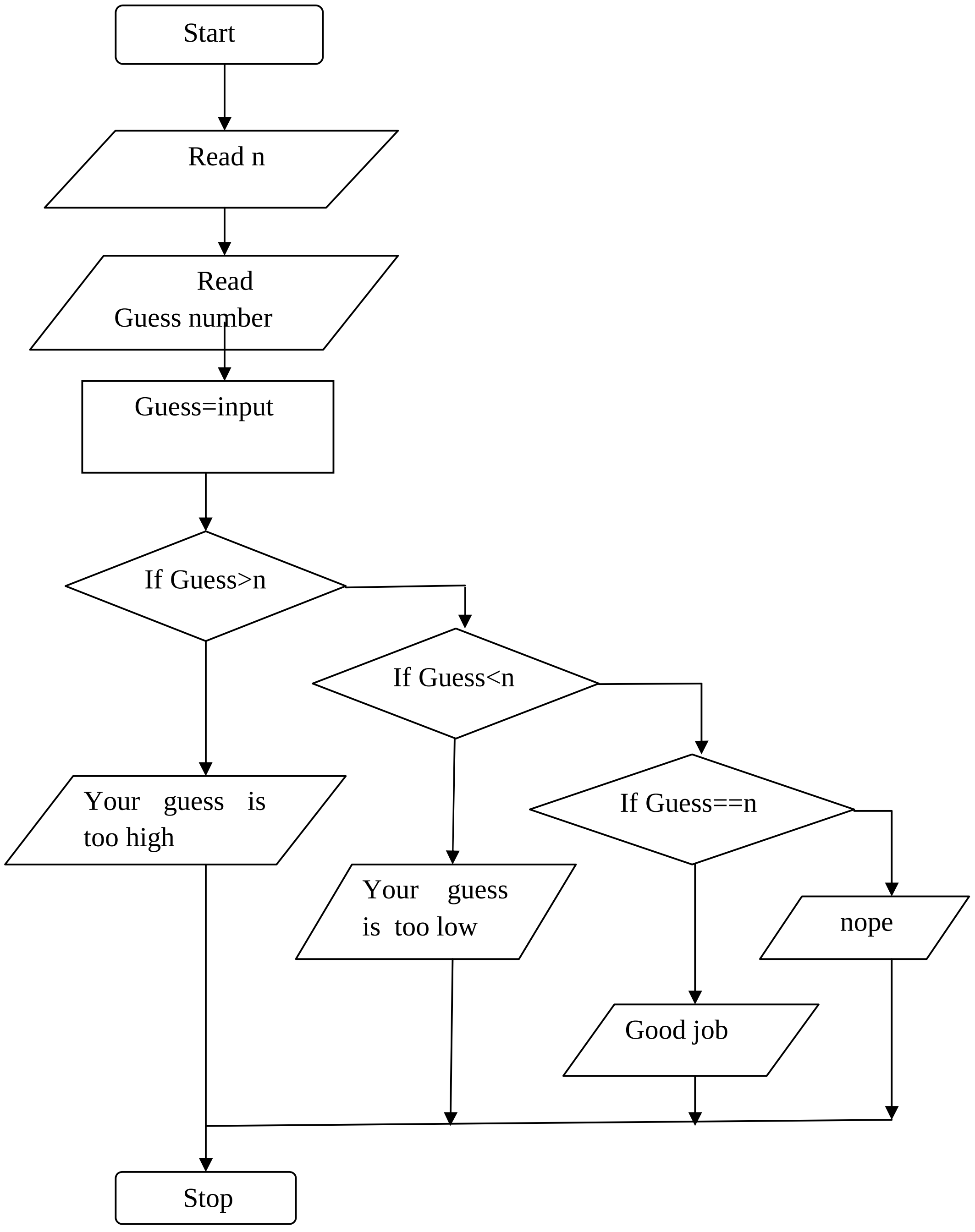
Algorithm:

```
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
```

Flowchart:


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
nope
Good job
Stop

2.Find minimum in a list

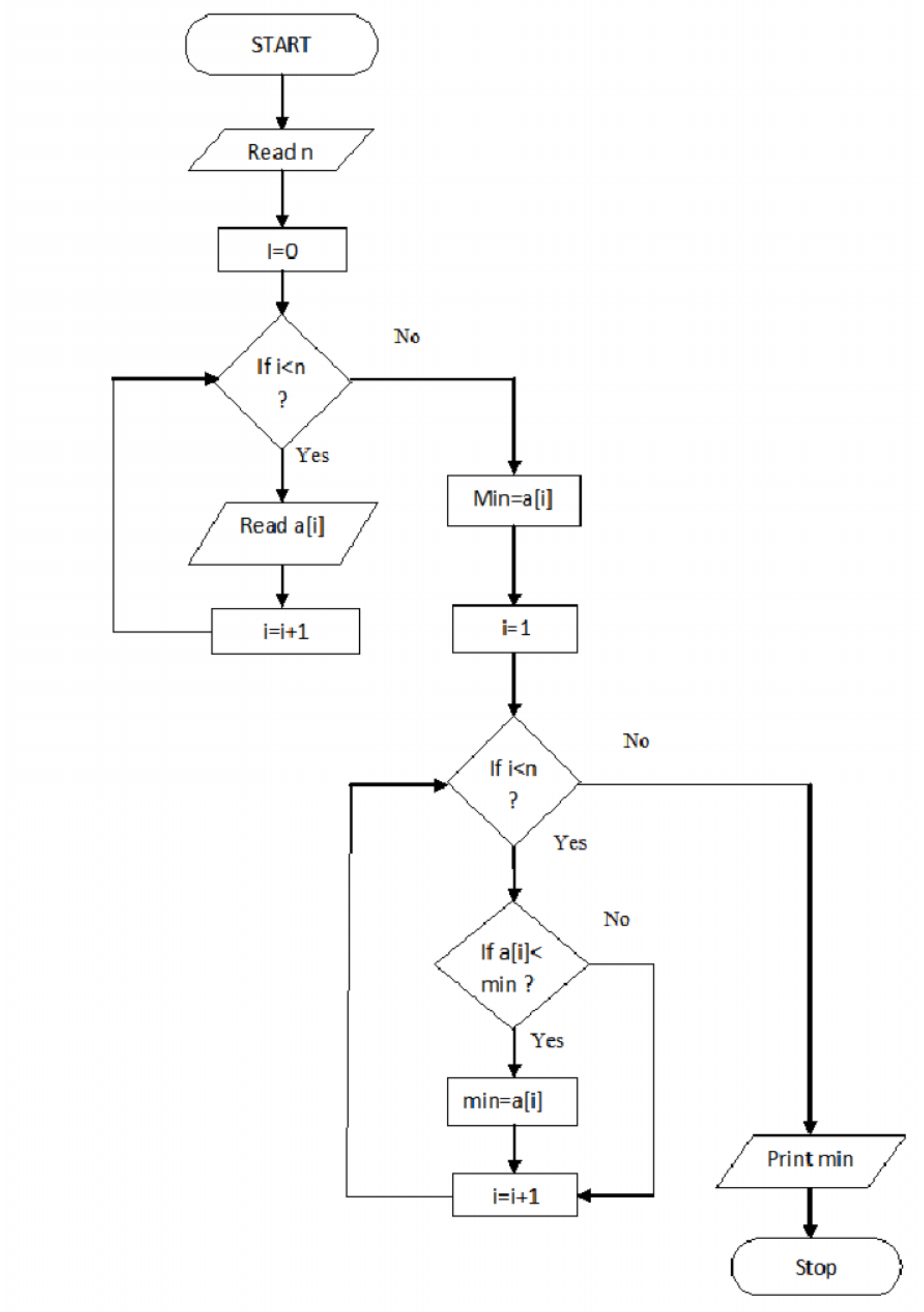
Algorithm:

```
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
```

Flowchart:


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

3. Insert a card in a list of sorted cards

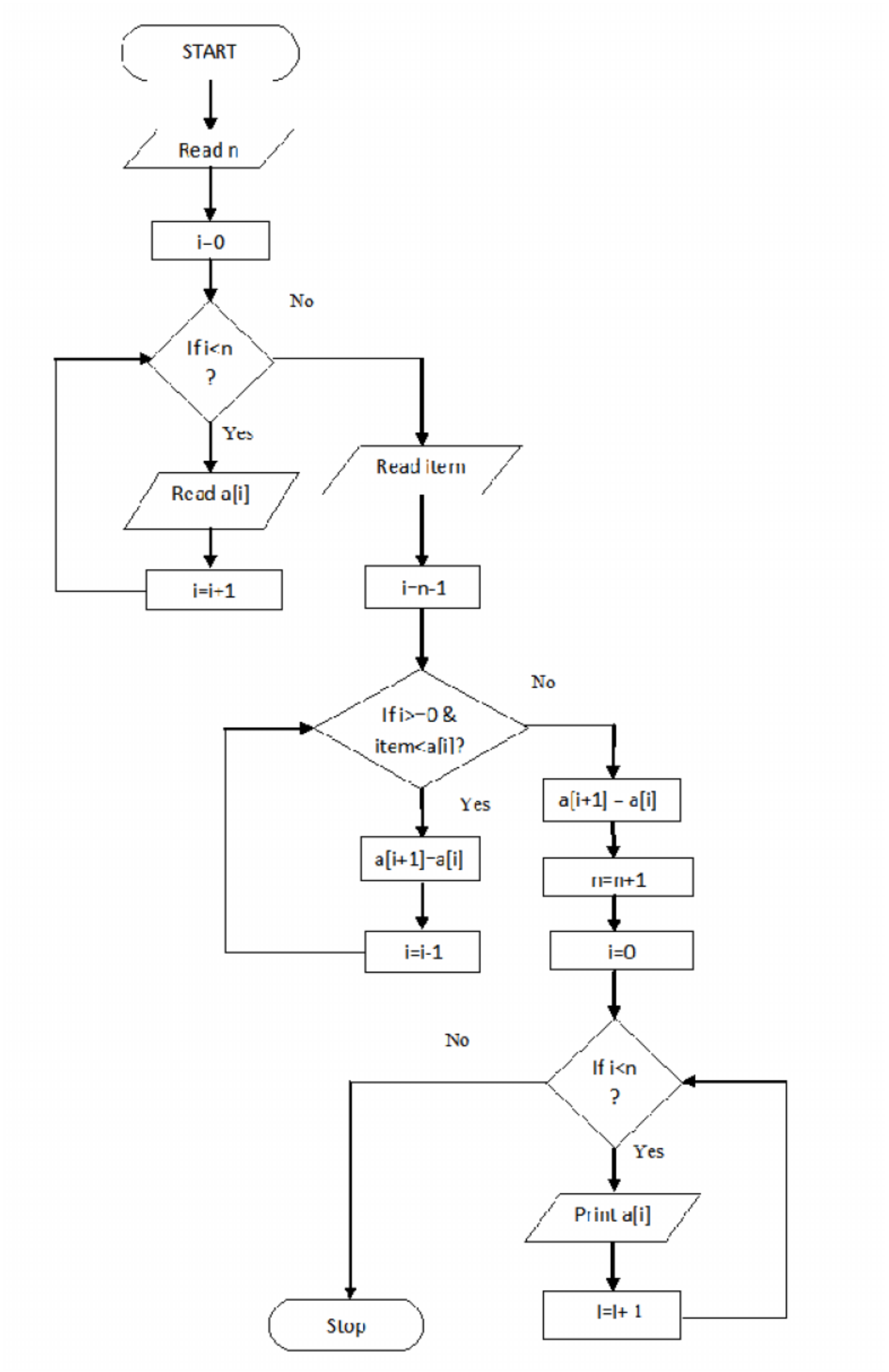
Algorithm:

```
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
```

Flowchart:


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

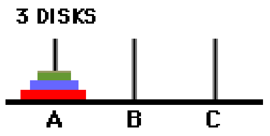
4.Tower of Hanoi

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.

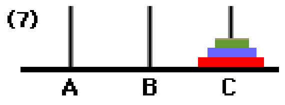
Pre-condition:

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 −

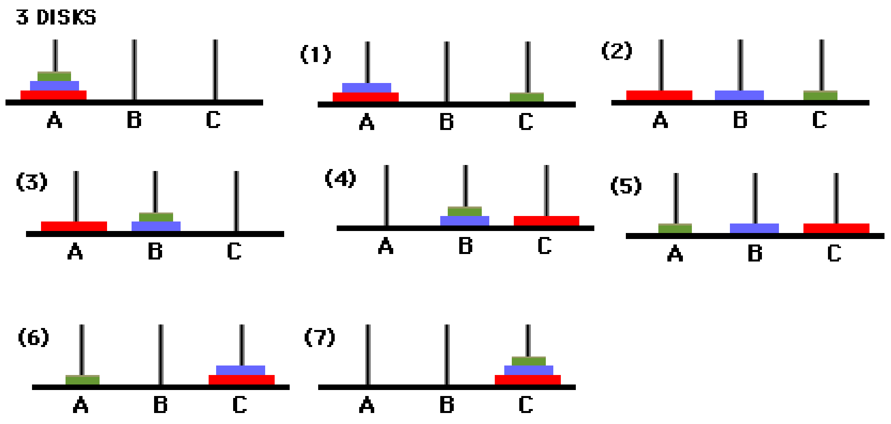
- First, we move the smaller (top) disk to aux peg.
- Then, we move the larger (bottom) disk to destination peg.
- And finally, we move the smaller disk from aux to destination peg.

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 –

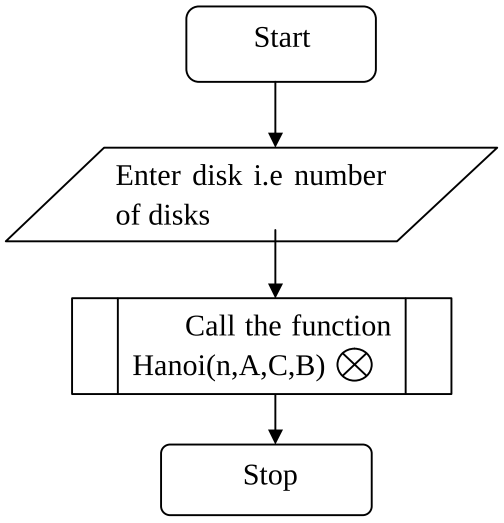
  Step 1 − Move n-1 disks from source to aux
  Step 2 − Move nth disk from source to dest
  Step 3 − Move n-1 disks from aux to dest

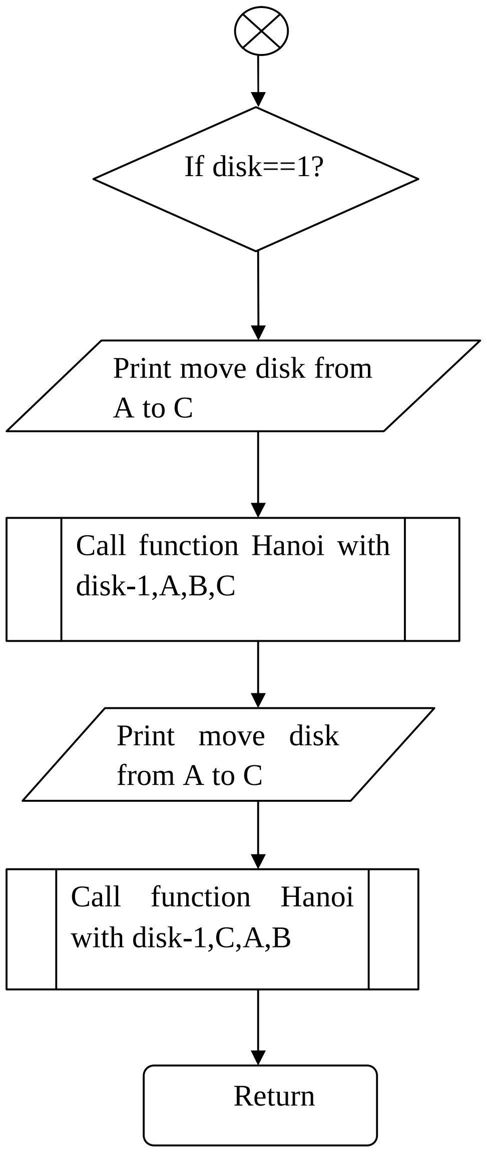


A recursive algorithm for Tower of Hanoi can be driven as follows –

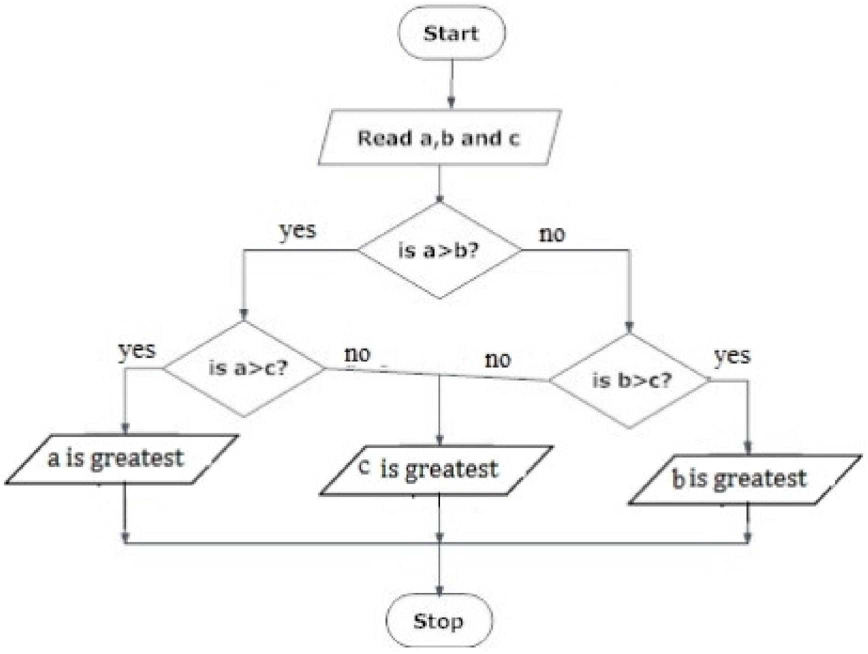
```
START
Procedure Hanoi(disk, source, dest, aux)
IF disk == 1, THEN
    move disk from source to dest
 ELSE
    Hanoi(disk - 1, source, aux, dest)     // Step 1
    move disk from source to dest          // Step 2
    Hanoi(disk - 1, aux, dest, source)     // Step 3
END IF
END Procedure
STOP
```

FLOW CHART

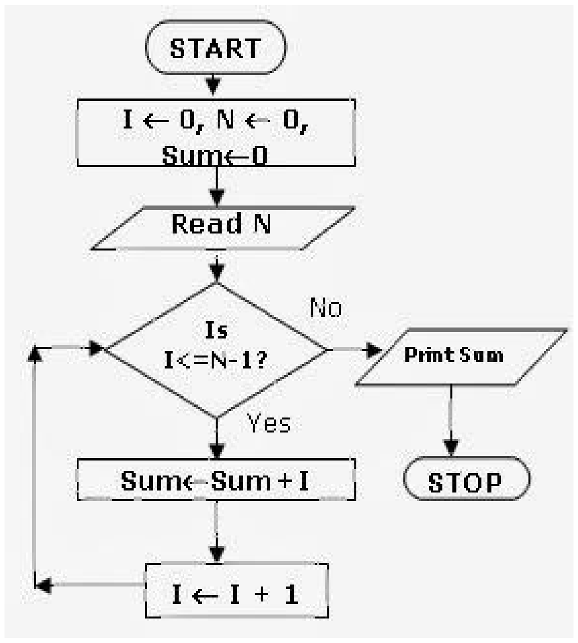

Start
Enter disk i.e number of disks
Call the function Hanoi(n,A,C,B) ⊗
Stop


⊗
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

5. Draw a flow chart to find greatest among three numbers.(AU 2018)



6. Draw a flow chart to find sum of n numbers(AU 2018)



2 MARKS

1. What is an algorithm?

   An algorithm is a finite number of clearly described, unambiguous do able steps that can be systematically followed to produce a desired results for given input in the given amount of time. In other word, an algorithm is a **step by step procedure to solve a problem** with finite number of steps.

2. What is Pseudo code?

   Pseudocode is an **informal high-level description** of the operating principle of a **computer program** or **algorithm**. **Pseudo means false** and **code** refers to **instructions written in programming language**.

3. What is Problem Solving?

   Problem solving is the systematic approach to define the problem and creating number of solutions. The problem solving process starts with the problem specifications and ends with a correct program.

4. Distinguish between algorithm and program.

	Algorithm	Program
1.	**Systematic logical approach** which is a well-defined, step-by-step procedure that allows a computer to solve a problem.	It is exact code written for problem **following all the rules of the programming language.**
2.	An algorithm is a finite number of clearly described, unambiguous do able steps that can be systematically followed to produce a desired results for given input in the given amount of time.	The program will accept the data to perform computation. **Program=Algorithm + Data**

5. Define Flow chart.

   A **graphical representation** of an algorithm. Flow charts is a diagram made up of boxes, diamonds, and other shapes, connected by arrows.

6. Write an algorithm to accept two numbers, compute the sum and print the result.

```
Step 1: Start
Step 2: Declare variables num1,num2 and sum,
Step 3: Read values num 1 and num2.
Step 4: Add  and assign the result to sum.
        Sum←num1+num2
Step 5: Display sum
```

7. Differentiate between iteration and recursion.

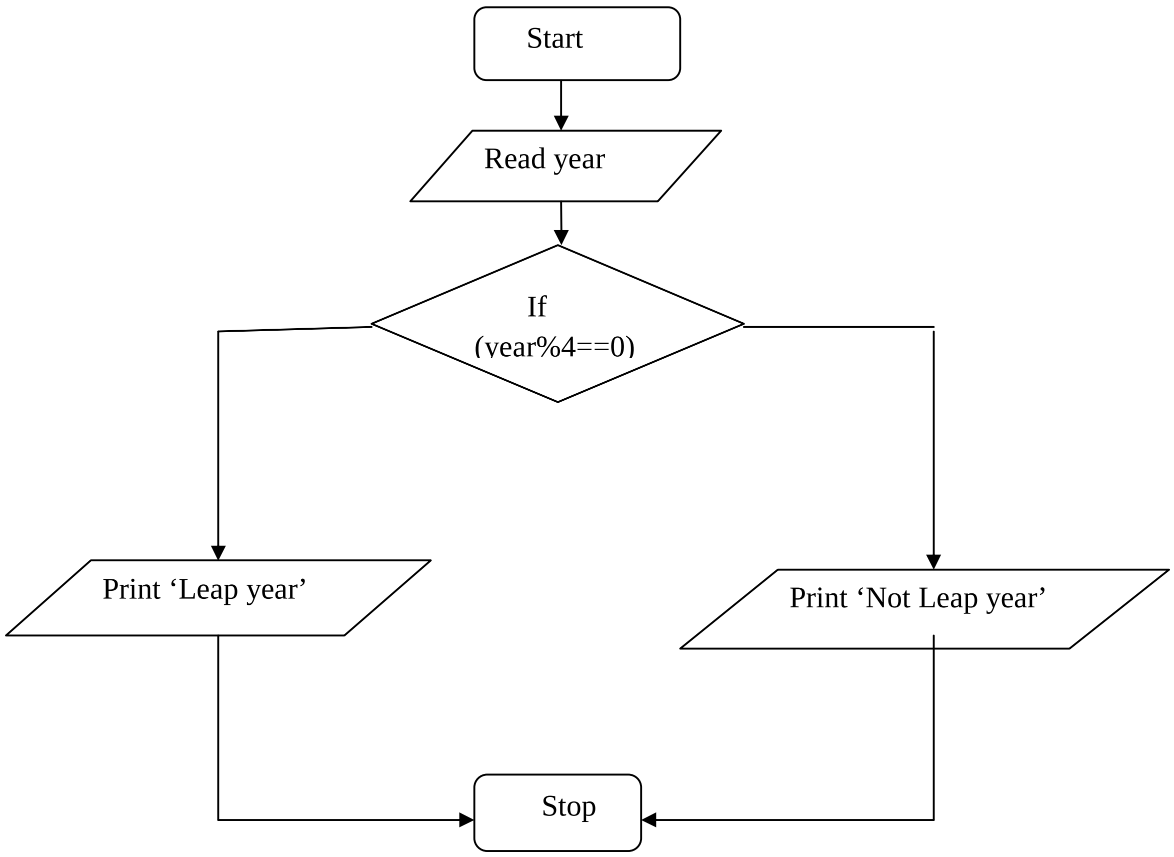
S.No	Iteration	Recursion
1.	Iteration is a process of executing certain set of instructions repeatedly, without calling the self function.	Iteration is a process of executing certain set of instructions repeatedly, by calling the self function repeatedly.
2.	Iterative methods are more efficient because of better execution speed.	Recursive methods are less efficient.
3.	It is simple to implement.	Recursive methods are complex to implement.

8. What is Programming language? With example.
   Programming Language is a formal language with set of instruction, to the computer to solve a problem. Java, C, C++, Python, PHP.

9. What are the steps for developing algorithms.
   • Problem definition
   • Development of a model
   • Specification of Algorithm
   • Designing an Algorithm
   • Checking the correctness of Algorithm
   • Analysis of Algorithm
   • Implementation of Algorithm
   • Program testing
   • Documentation Preparation

10. What are the Guidelines for writing pseudo code?
- Write one statement per line
- Capitalize initial keyword
- Indent to hierarchy
- End multiline structure
- Keep statements language independent.

11.Draw a flow chart to find whether the given number is leap year or not.


Start
Read year
If
(year%4==0)
Print 'Leap year'
Print 'Not Leap year'
Stop