UNIT I (GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING)

ALGORITHMIC PROBLEM SOLVING

Algorithms are procedural solutions to problems. Algorithmic problem solving is defined as the formulation and solution of problems where solution involves the principles and techniques to construct the correct algorithms. It means that solving problems that require formulation of an algorithm for their solution.

Steps in designing and analyzing an algorithm

The steps are

i) Understanding the problem

ii) Ascertaining the capabilities of a computational device

iii) Choosing between exact and approximate problem solving

iv) Deciding on appropriate data structures.

v) Algorithm design techniques

vi) Methods of specifying an algorithm.

vii) Proving an algorithm's correctness.

viii) Analyzing an algorithm.

ix) Coding an algorithm.

i) Understanding the problem

→Before designing an algorithm, you need to understand the problem completely.

→Read the problem description carefully.

→ Check if it is similar to some standard problems and if a known algorithm exists.

→Ask Questions if you have any doubts and do a few small examples by hand.

→Think about special cases.

→Ask Questions again if needed.

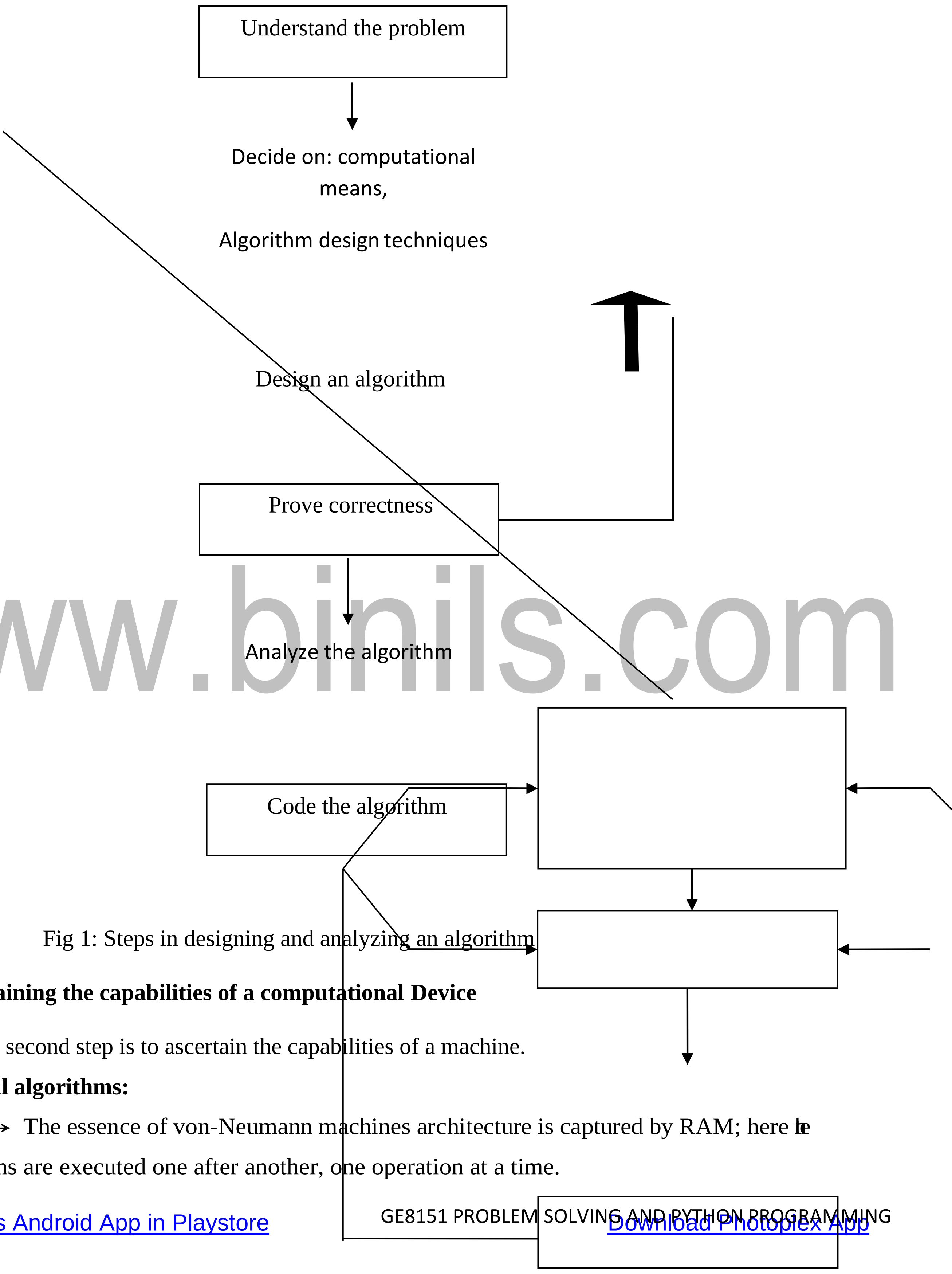


Fig 1: Steps in designing and analyzing an algorithm

ii) Ascertaining the capabilities of a computational Device

The second step is to ascertain the capabilities of a machine.

Sequential algorithms:

⟶ The essence of von-Neumann machines architecture is captured by RAM; here the instructions are executed one after another, one operation at a time.

→Algorithms designed to be executed on such machines are called sequential algorithms.

Parallel algorithms:

→ An algorithm which has the capability of executing the operations concurrently iscalled parallel algorithms.

→RAM model doesn't support this.

iii) Choosing between exact and approximate problem solving

→The next decision is to choose between solving the problem exactly or solving i approximately.

→Based on this, the algorithms are classified as exact and approximation algorithms.

Exact algorithms:

→ The algorithms that can solve the problem exactly are called exact algorithms.

Approximate algorithms:

→The algorithms that can solve the problem not exactly are called approximate algorithms.

→There are three issues to choose an approximation algorithm.

→First, there are certain problems like extracting square roots, solving non-linear equations which cannot be solved exactly.

→Secondly, if the problem is complicated it slows the operations. E.g. Traveling salesman problem.

→Third, this algorithm can be a part of a more sophisticated algorithm that solves aproblem exactly.

iv) Deciding on appropriate data structures

→Data structures play a vital role in designing and analyzing the algorithms.

→Some of the algorithm design techniques also depend on the structuring data specifying a problem's instance.

Algorithm + Data structure = Programs

v) Algorithm design techniques

→An algorithm design technique 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 important for two reasons.

i) First, they provide guidance for designing for new problems.

ii) Second, algorithms are the cornerstones of computer science.

vi) Methods of specifying an algorithm

Once you have designed an algorithm, you need to specify it in some fashion. There are two options for specifying algorithms.

⟶ Pseudo code

⟶ Flowchart

Pseudo code is a mixture of a natural language and programming language. It is more precise.

Flowchart is a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.

vii) Proving an algorithm's correctness

→Once an algorithm has been specified, you have to prove its correctness.

→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.

viii) Analyzing an algorithm

⟶ Our algorithm should possess several qualities.

⟶ Analysis of algorithms and performance analysis refers to the task of determining how much computing time and storagean algorithm requires.

→After correctness, the most important quality of an algorithm is Efficiency.

→There are two kinds of algorithm Efficiency.

i) **Time efficiency**: It indicates how fast the algorithm runs.

ii) **Space efficiency:** Indicates how much extra memory the algorithm needs.

ix) Coding an algorithm

⟶ Algorithms are implemented as computer programs.

⟶ Verification and validation is done for error correction.

UNIT I (GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING)

BUILDING BLOCKS OF AN ALGORITHM

Algorithm is defined as the effective step-by-step procedure to solve the problem in a finite number of steps. Algorithms can be designed using the following components. They are

i) Instructions/Statements

ii) State

iii) Control Flow

iv) Functions

Instructions/Statements

Instructions/Statements are the steps which solves a particular problem. Most algorithms have the basic parts of statements.

→Description of the problem.

⟶ Setup

⟶ Parameters

⟶ Execution

⟶ Conclusion

Example: To find the sum of two numbers.

Description of the problem:

To find the sum of two numbers.

Setup:

Two numbers required for addition and one variable for storing the result.

Parameters:

- Read first number.
- Read the second number.

Execution:

Calculate the sum of two numbers (a,b) result = a + b

Conclusion:

The desired output is the sum of two numbers which is stored in the variable 'result'.

State

State is defined as the condition of algorithm at a moment in a time. Algorithm can be in any of the following states.

1. START state
2. INPUT state
3. PROCESS state
4. OUTPUT state
5. STOP state

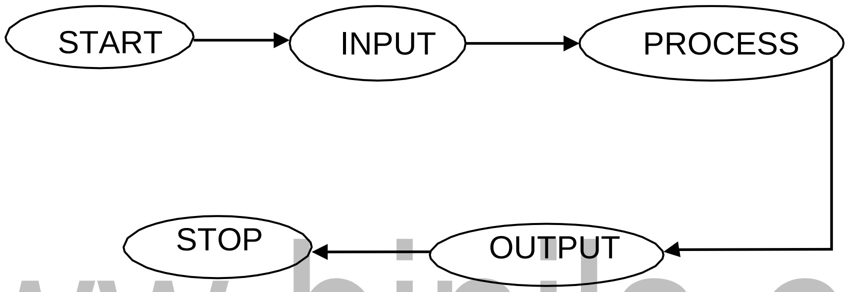


Fig 1: States of an algorithm

1. **START state:**

   It represents the starting of the program.

2. **INPUT state:**

   It represents the process of reading the input from the user.

3. **PROCESS state:**

   It represents the process done in the program. E.g. Addition, Subtraction etc..

4. **OUTPUT state:**

   It represents the output displayed on the screen.

5. **STOP state:**

   It represents the ending of the program.

Example: Algorithm to find the sum of two numbers.

a. Start (START state)
b. Read A, B (INPUT state)
c. C=A+B (PROCESS state)
d. Print C (OUTPUT state)

e. Stop (STOP state)

Control flow

It represents the order of statement execution and direction of flow of programs.

There are three types of control flow.

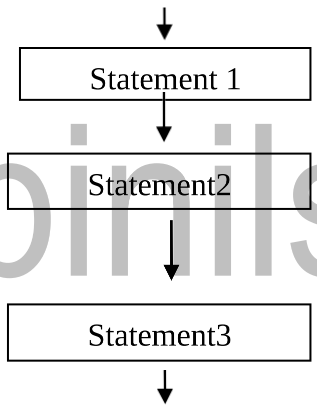
i) Sequential control flow.

ii) Selection control flow.

iii) Repetition control flow.

i) Sequential control flow

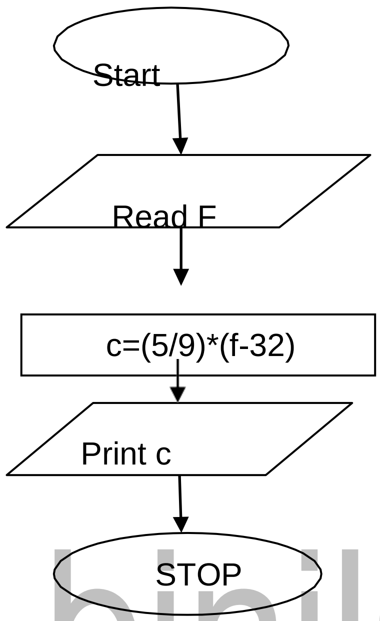
The steps of an algorithm are carried out in a sequential manner.



Example: Temperature conversion

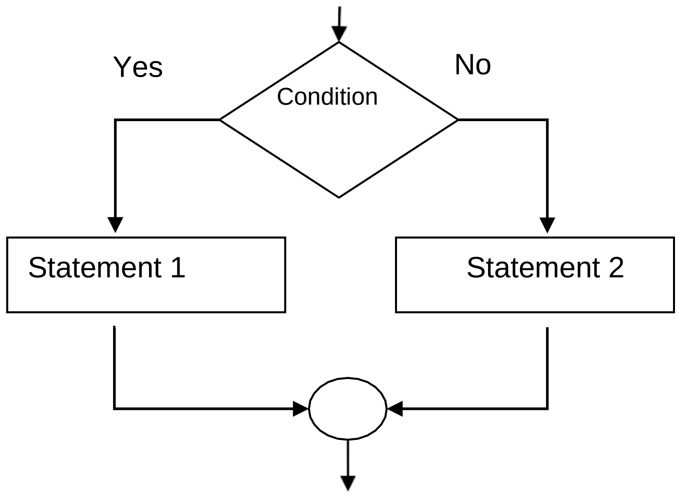
1. Start
2. Read temperature in fahrenheit f.
3. c = (5/9)*(f-32)
4. Print c
5. Stop

Flowchart:



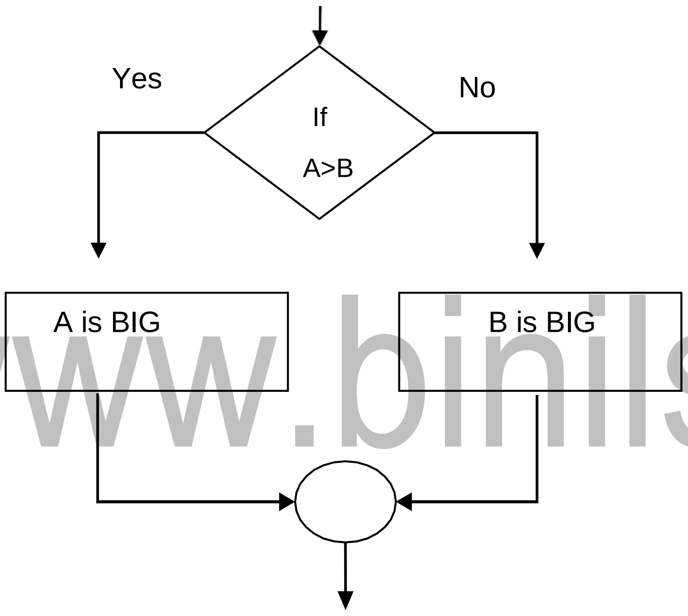
ii) Selection control flow

Only one of the alternative steps is executed based on the condition.



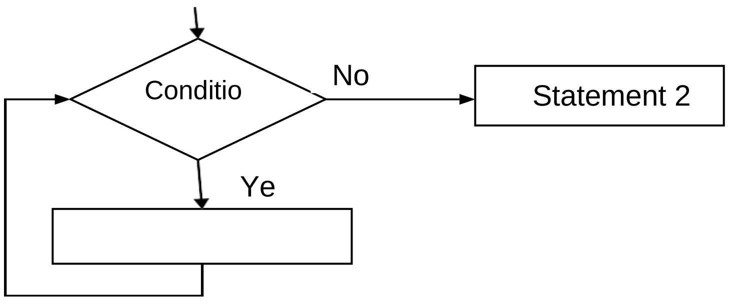
Example: Algorithm to find the biggest among two numbers.

1. Start
2. Read a, b
3. If a>b then Print "a is big"
4. Else Print "b is big"
5. Stop



iii) Repetition control flow

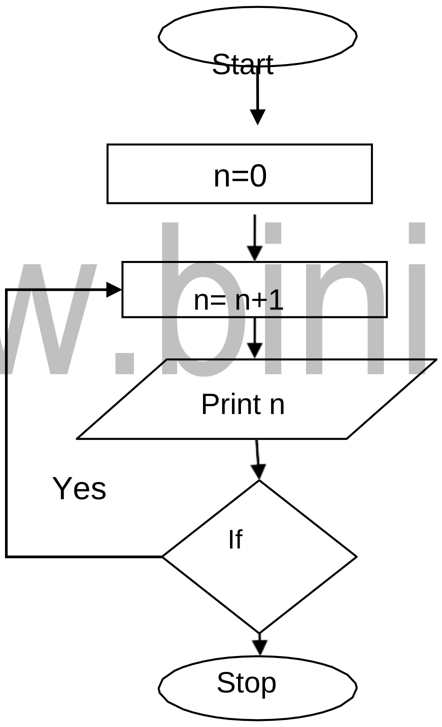
One or more steps are performed repeatedly. This logic is used for producing loops in the program. The looping process can either be one time or multiple times based on the condition.



Example: Algorithm to print numbers from 1 to 10.

1. Start
2. Initialize n=0
3. Increment n=n+1
4. Print n
5. If n<=10 then goto step 3
6. Else goto step 7
7. Stop

Flowchart:



Functions

→A function is a block of organized, reusable code which is used to perform a task or action.

→It provides better modularity.

→They are also called as methods, routines, procedures etc.

→"Add" is a function and it performs addition operation.

Example: Algorithm to find the biggest among N numbers**.**

```
1. Start
2. Read l1, l2, l3…ln into array.
3. Initialize max=l1
4. Read next number l1, do the following
      i. If max< l1 then
            I) max = l1
      ii. Repeat step" i" until n
5. Print max
6. Stop
```

Advantages of algorithm

- Easy to understand.
- Easy to debug.
- It is easy for a programmer to convert algorithm into actual program.

Disadvantages of algorithm

- It is time consuming. (An algorithm is developed first, which is then converted into flow chart and then program).
- It is difficult to show branching and looping.

UNIT I (GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING)

ILLUSTRATIVE PROBLEMS:

1. Guess an integer number in a range

Guessing game – guessing a number within a range of numbers.

Algorithm:

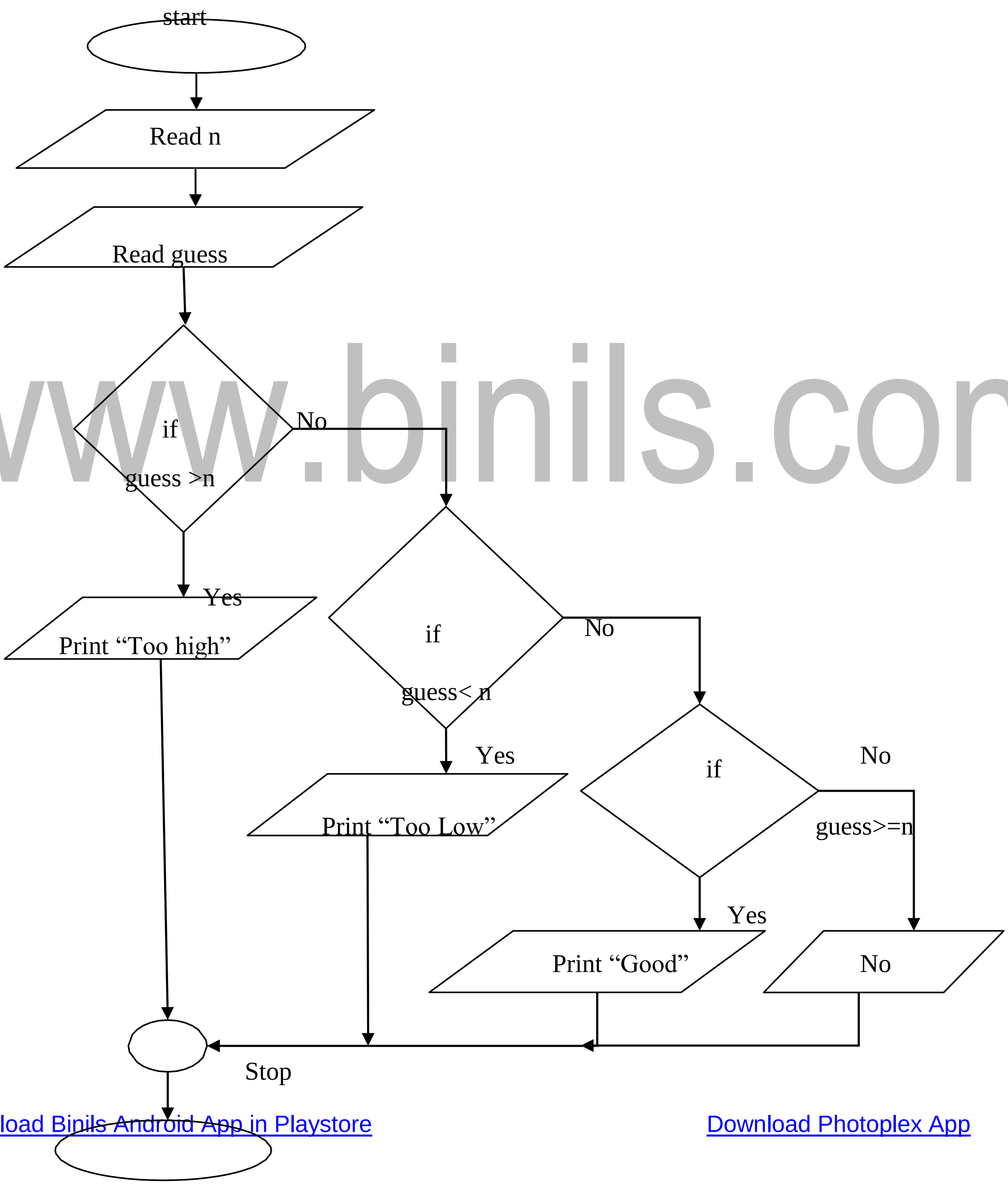
```
1. Start
2. Read n.
3. Read a guess number.
4. If guess > n
5. Print “Your guess is too high”
6. If guess < n
7. Print “ Your guess is too low”
8. Else
9. If guess = n
10. Print “Good job” else print “no”
11. Stop
```

Pseudo code:

```
BEGIN
READ n
OBTAIN guess
IF guess > n
        DISPLAY “Your guess is too high”
IF guess < n
        DISPLAY “Your guess is too low”
ELSE
IF guess=n
        DISPLAY “Good Job” ELSE DISPLAY “No’
ENDIF
```

```
        ENDIF
        ENDIF
        END
```

Flowchart:



2. Tower's of Hanoi

A Tower's of Hanoi is a children's playing game, played with three poles and a number of different sized disks which is stacked on the poles. The disks are stacked on the left most pole in ascending order initially. ie) The largest on the bottom and the smallest on the top.

Rules to be followed:

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

ii) Only the "top" disk can be removed.

iii) No large disk can sit over a small disk.

The objective is to transfer the disks from the left most pole to the right most pole by using the centre pole as the temporary pole.

The steps are

i) Move the top n-1 disks from the left pole to the centre pole, n is the number of disks.

ii) Move the nth largest disk to the right pole.

iii) Move the n-1 disks on the centre pole to the right pole.

The total number of moves will be $2^n - 1$, where n is the number of disks.

Algorithm:

```
1. Start
2. Read disk, source, dest, aux
3. Call the function Hanoi(disk, source, dest, aux)
4. Stop
```

Algorithm for function Hanoi(disk, source, dest, aux):

```
1. Start
2. If disk=1
        Move disk from source to dest
3. Else
        Hanoi(disk-1, source, aux, dest)
        Move disk from source to dest
        Hanoi(disk-1, aux, dest, source)
4. Return
```

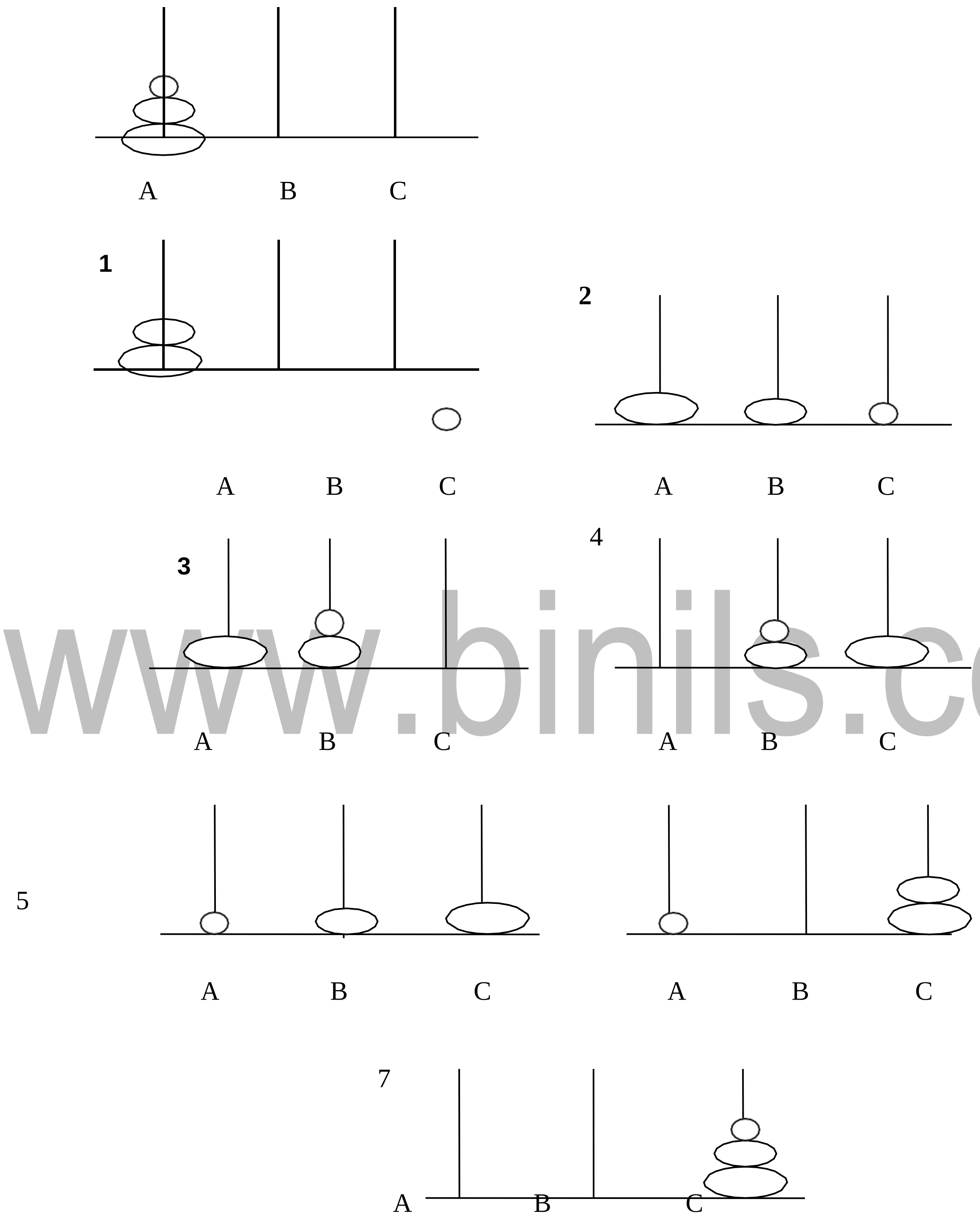
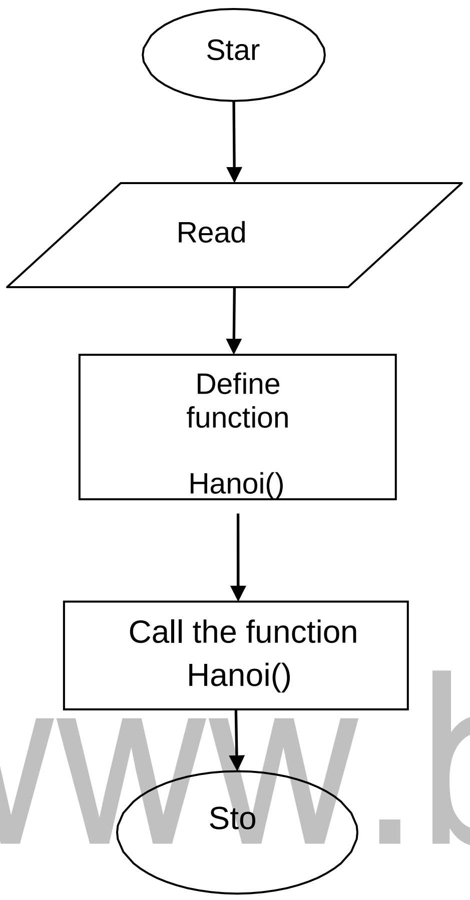
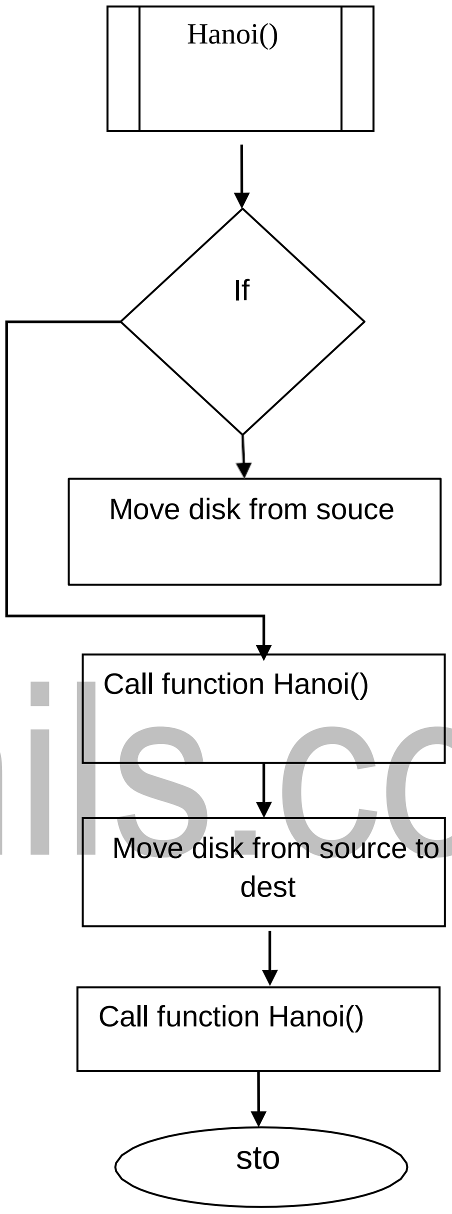


Fig 1: Tower's of Hanoi with 3 disks

Flowchart:


Star
Read
Define function Hanoi()
Call the function Hanoi()
Sto


Hanoi()
If
Move disk from souce
Call function Hanoi()
Move disk from source to dest
Call function Hanoi()
sto

Pseudo code:

```
BEGIN
READ disk, source, dest, aux
FUNCTION Hanoi (disk, source, dest, aux)
END
```

Pseudo code for function Hanoi (disk, source, dest, aux)

```
BEGIN
 IF disk=1 THEN
         Move disk from source to dest
 ELSE
         Hanoi (disk-1, source, aux, dest)
         Move disk from source to dest
         Hanoi (disk-1, aux, dest, source)
ENDIF
END
```

UNIT I (GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING)

ILLUSTRATIVE PROBLEMS:

1. Finding minimum in a list

Problem description:

The problem is to find the minimum element in the given list of elements. Finding minimum in a list of elements can be achieved in different ways. One way is to sort the list of elements in ascending order and get the first element as minimum. Another method is to compare each element with other. As an initial step, first element of the list is considered as minimum element. And in each iteration, each element in the list is compared with the minimum. If the element in the list is less than the minimum then swap both elements else compare with the next element in the list. These steps are continued until the end of the list and finally print the minimum.

Algorithm:

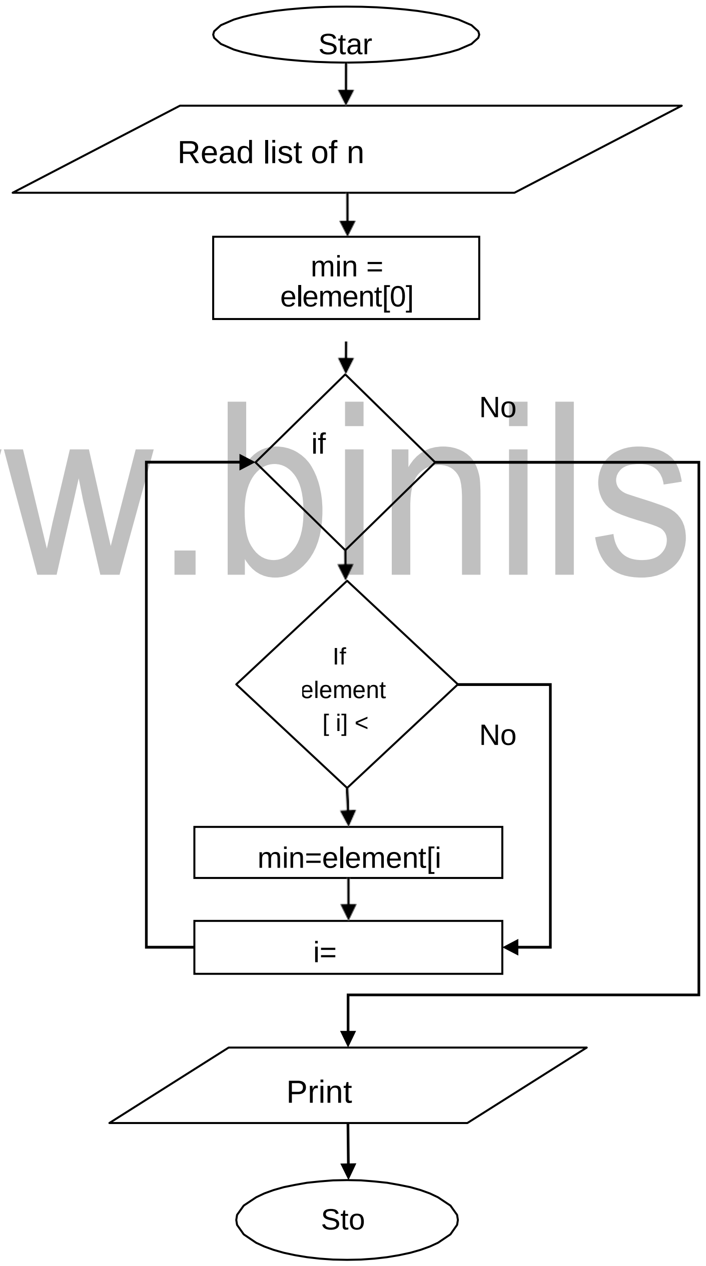
```
1. Start.
2. Read the list of elements.
3. Set first element in the list as minimum.
4. Move to the next element in the list.
5. If current element<minimum then
       Set minimum =current element.
6. Repeat Step 4 and Step 5 until the end of the list is reached.
7. Print the minimum value in the list
8. Stop
```

Pseudo code:

```
BEGIN
READ the list of elements.
INITIALIZE min with the first element of the list
FOR each element in the list of elements
    IF element is less than min
        COPY element to min
```

```
          ENDIF
    ENDFOR
   PRINT min as the minimum value
    END
```

Flowchart:



2. To insert a card in a list of sorted card

Playing cards are one of the techniques of sorting and the steps are shown as follows: Start with an empty left hand and cards face down on the table. Then remove one card at a time from the table and Insert it into the correct position in the left hand. To find a correct position for a card, we compare it with each of the cards already in the hand from left to right. Once the position is found, the cards from that position are moved to the next higher indexed position and in that order. New card is inserted at the current position.

Algorithm:

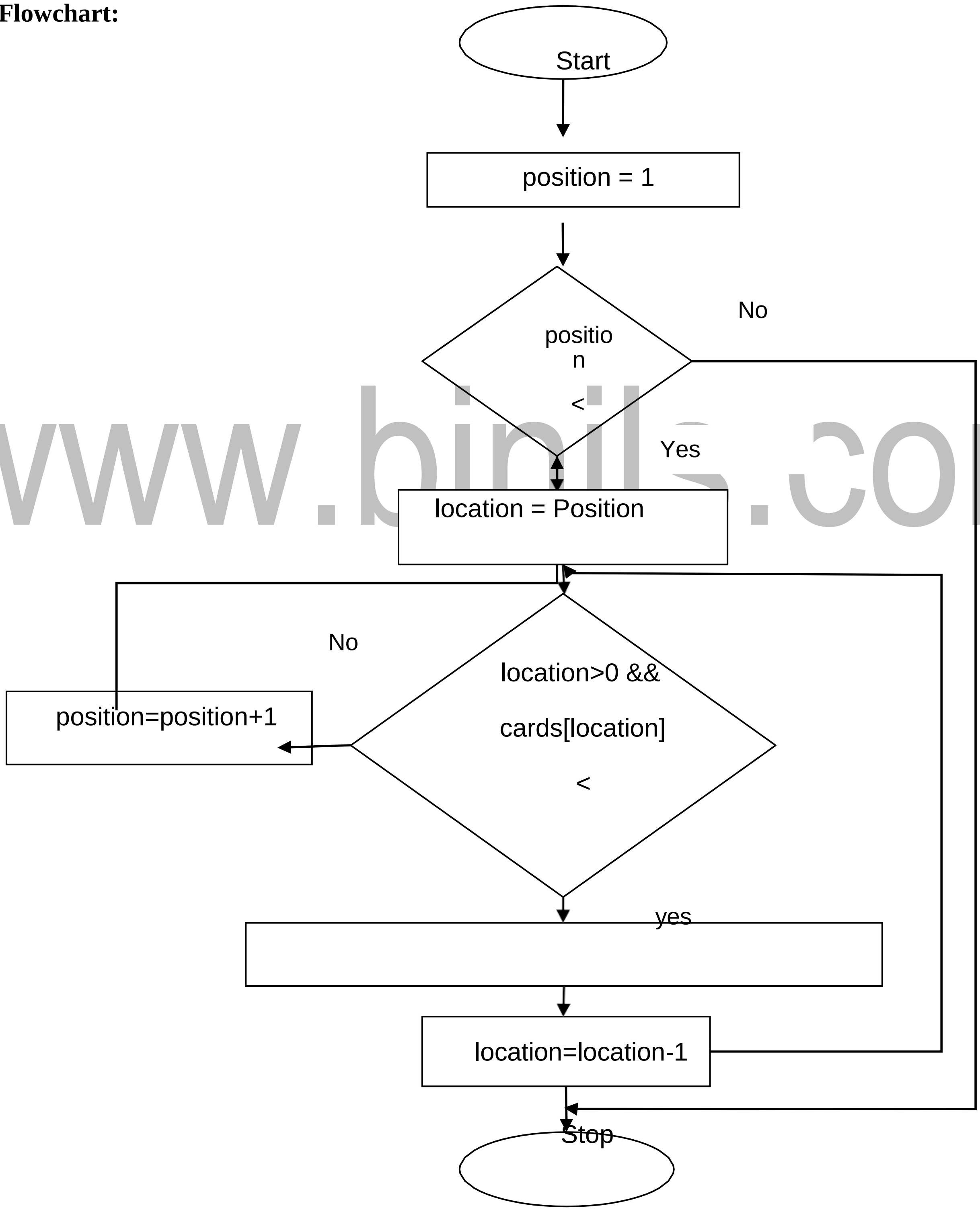
```
1. Start
2. Read the list of sorted cards.
3. Read the new card.
4. Insert a new card to the end of the list.
5. For i=1 to len(a)
6. current_card=a[i]
7. pos=i
8. While pos>0 and a[pos-1]>current_card
9. a[pos]=a[pos-1]
10. pos=pos-1
11. a[pos]=currentcard
12. Print sorted list of cards.
13. Stop
```

Pseudo code:

```
BEGIN
READ the list of sorted cards.
READ the new card.
ADD new card to the end of the sorted list.
FOR i=1 to len(a)
        current_card=a[i]
        pos=i
```

```
WHILE pos>0 and a[pos-1]>current_card
          a[pos]=a[pos-1]
          a[pos]=current_card
ENDWHILE
 PRINT sorted card list.
END
```

Flowchart:



UNIT I (GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING)

NOTATION

Pseudo code

Pseudo code is made up of two words: Pseudo and code. Pseudo means 'imitation' and 'code' refers to instructions written in programming language. Pseudo code is not a real programming language, but it looks like a programming language. Pseudo code is also called as "Program Design Language [PDL]". It is an outline of a program, written in a form that can be easily converted into real programming statements. Pseudo code instructions are written in normal English.

Rules for writing pseudo code:

i) Write one statement per line.

ii) Capitalize the keywords.

iii) End Multiline structure.

iv) Keep Statements language independent.

v) Intend to show hierarchy.

Keywords used in pseudo code:

START: BEGIN

INPUT: READ, OBTAIN, GET, INPUT, DEFINE

OUTPUT: OUTPUT, PRINT, DISPLAY, SHOW

COMPUTE:CALCULATE, COMPUTE, ADD, SUBTRACT, INITIALISE, DETERMINE

INITIALIZE: SET, INITIALIZE

ADD ONE: INCREMENT

STOP: END

Pseudo code guidelines:

i) Pseudo code statements should be written in simple English.

ii) Each statement should be written in separate line.

iii) The keywords should be capitalized.

iv) Pseudo code should be programming language independent.

v) The steps must be understandable.

vi) Each set of instructions are written from top to bottom.

Advantages (Benefits):

⟶ It can be read and understood easily.

⟶ It can be done easily on a word processor.

⟶It can be modified easily.

⟶It occupies less space.

⟶ It will not run over manypages.

⟶ Converting a pseudo code to a program is simple.

Disadvantages (Limitations):

⟶It is not visual.

⟶ We do not get a picture of the design.

⟶ There is no standardized style or format.

⟶ For a beginner, it is more difficult to follow the logic or write pseudo code.

Example 1: Write Pseudo code to calculate sum and average for n numbers.

```
BEGIN
INITIALIZE sum=0, i=1
READ n
FOR i < = n, then
        COMPUTE sum = sum +i
        CALCULATE i=i+1
END FOR
COMPUTE avg = sum/n
PRINT sum, avg
END
```

Example 2: Write Pseudo code to add two numbers**.**

```
BEGIN
SET C=0
READ A, B
ADD C=A+B
PRINT C
```

```
END
```

Example 3: Write Pseudo code to calculate area of circle.

```
BEGIN
READ radius r
INITIALIZE pi=3.14
CALCULATE Area=pi * r *r
PRINT Area
END
```

Example 4: Write Pseudo code to read number n and print the integers counting up to n.

```
BEGIN
READ n
INITIALIZE i to 1
FOR i <= n, then
    DISPLAY i
    INCREMENT i
END FOR
END
```

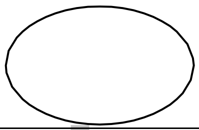
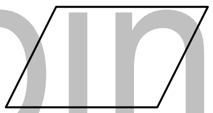
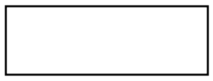
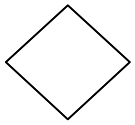
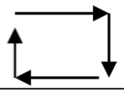
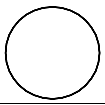
Example 5: Write Pseudo code to find the greatest among two numbers.

```
BEGIN
Read A, B
IF A >B
    PRINT "A is greatest"
ELSE
    PRINT "B is greatest"
ENDIF
END
```

Flowchart

Flowchart is a diagrammatic representation of an algorithm. A flowchart is a picture of the separate steps of a process in sequential order. Flowchart is made up of boxes, diamonds and other shapes connected by arrows where each shape represents a step in the process. The arrows show the order of the flow of execution. Flowcharts are used in designing or documenting a process or program. The logic of the program is communicated in a much better way by using a flowchart.

Flowchart symbols:

Sl.No	Name of the symbol	Symbol	Description
1	Start/Stop		Represent the start and stop of the program
2	Input/Output		Denoted either an input or output operation
3	Process		Denotes the process to be carried out
4	Decision		Represent decision making and branching
5	Flow lines		Represents the sequence of steps and direction of flow
6	Connector		Connects remote parts of the flowchart on the same page

Guidelines for drawing flowchart:

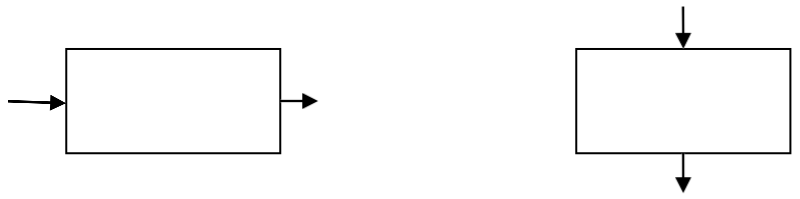
i) All necessary requirements should be listed out in logical order.

ii) There should be **START** and **STOP** in the flowchart.

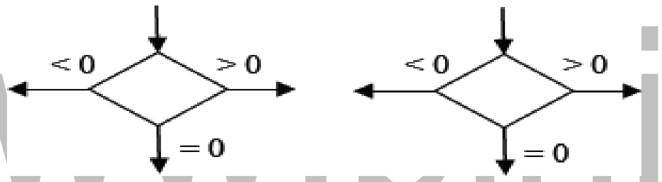
iii) The flowchart should be clear, neat and easy to follow.

iv) The direction of flow is from left to right or top to bottom.

v) Only one flow line should emerge from a process symbol.



vi) Only one flow line should enter a decision symbol but 2 or 3 flow line can leave the decision symbol.



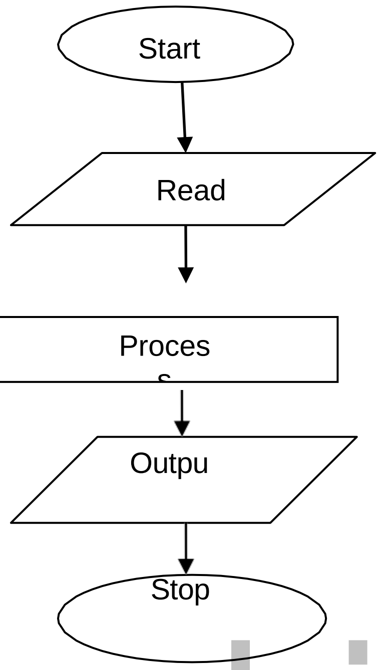
vii) Only one flow line is used with terminal symbol.

viii) If the flowchart becomes complex, connector symbols are used to reduce the number of flow lines.

ix) The text within the symbols should be brief.

x) The validity of flowchart should be tested by passing simple test data.

Basic design structure of flowchart:



Advantages:

i) Better communication

It is easy for the programmer to explain the logic of program.

ii) Effective analysis

It is easy to analyze the problem effectively.

iii) Proper documentation

With the help of flowchart good documentation is done for various purposes.

iv) Efficient coding

Flowchart acts as a guide during the system analysis and program development phase.

v) Efficient debugging

It helps in debugging process.

vi) Efficient program maintenance

The maintenance of a program becomes easy with the help of the flowchart.

Disadvantages (Limitations):

i) Complex logic

Sometimes the logic of the program is quite complicated. In such a case flowcharts become complex.

ii) Alterations and modifications

If alterations are required, the flowchart needs to be redrawn completely.

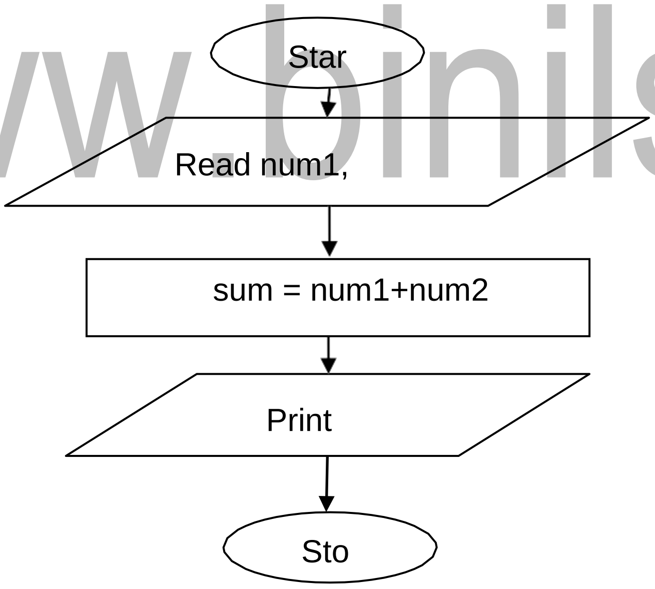
iii) Reproduction

Reproduction of the flowchart becomes a problem because it cannot be typed.

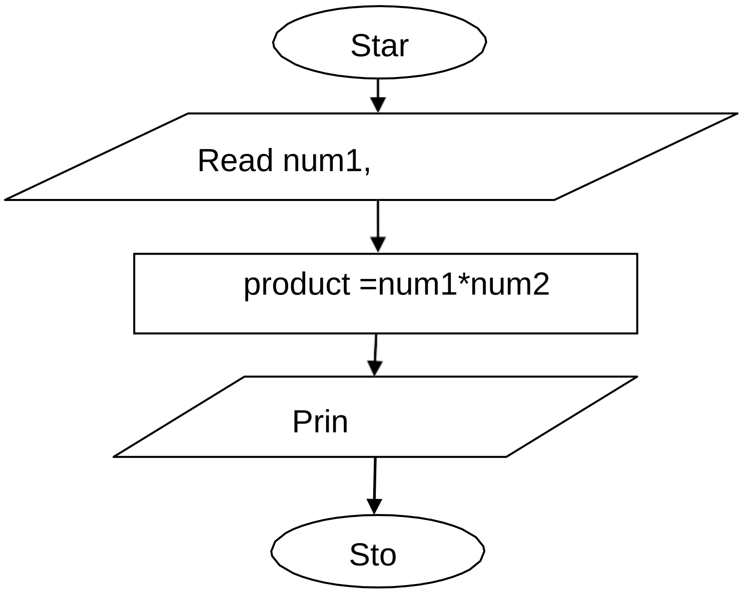
iv) Cost

High cost for large applications.

Example1: Draw a flowchart to add two numbers.



Example2: Draw a flowchart to find the product of two numbers.



UNIT I (GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING)

Programming languages

Programming languages are used to communicate instructions to the computer.

Programming languages are written based on syntactic and semantic rules.

Programming languages can be divided into nine major categories.

i) Interpreted programming language

ii) Functional programming language

iii) Compiled programming language

iv) Procedural programming language

v) Scripting programming language

vi) Markup programming language

vii) Logic-based programming language

viii) Concurrent programming language

ix) Object-Oriented programming language

i) Interpreted programming language

An interpreted programming language is a programming language that executes instructions directly, without compiling a program into machine-language instructions.

Example: BASIC (Beginners All Purpose Symbolic Instruction Code), LISP (List Processing Language), Python.

ii) Functional programming language

Functional Programming languages define every computation as a mathematical evaluation. They are based on mathematical functions. They focus on application of functions.

Example: Clean, curry, F#, Haskell and Q

iii) Compiled programming language

Compiled Programming language is a programming language which uses compilers to convert the source code into machine code.

Example: Ada, algol, C,C++,C#, COBOL, Java, Fortran, VB

iv) Procedural programming language

A procedural language is a type of computer programming language that specifies a series of well-structured steps and procedures within its programming context to compose a program. These languages specify the steps that the program should take to reach to an intended state. Procedure is a group of statements. It makes the program structured and helps to reuse the code.

Example: CLIST, Hypertalk, MATLAB, PL/I

v) Scripting programming language

Scripting languages are Programming languages that control applications.

Example: AppleScript, AWK, MAYA, PHP, VBSCRIPT.

vi) Markup programming language

Markup language is an artificial language that define how the text to be displayed.

Example: CURL, SGML, HTML, XML, XHTML.

vii) Logic-based programming language

Logic-based programming language is a type of programming language that is based on logic.

Example: ALF, Fril, Janus, Leda, Prolog.

viii) Concurrent programming language

It is a computer programming language that executes the operations concurrently.

Example: ABCL, Afnix, E, Joule, Limbo.

ix) Object-Oriented programming language

It is a Programming languages based on concepts of objects. Objects contain data and functions.

Example: Agora, Beta, Lava, Moto, Scala, Slate.

Machine language:

In machine language, instructions are in the form of 0's and 1's. Instructions in machine language consist of two parts.

OPCODE	OPERAND

⟶ OPCODE tells the computer what functions are to be performed.

⟶ OPERAND tells the computer where to store the data.

Assembly language:

In assembly language, mnemonic code is assigned to each machine language instruction which is easy to remember and write. Instruction in assembly language consists of 4 parts.

LABEL	OPCODE	OPERAND	COMMENT

4GL language:

4GL (Fourth generation languages) are simple which is used to access databases.

UNIT I (GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING)

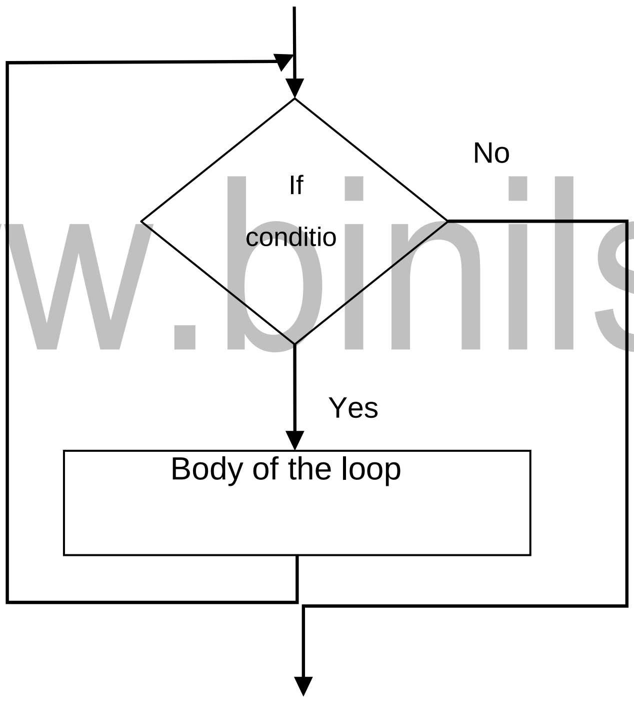
SIMPLE STRATEGIES FOR DEVELOPING ALGORITHMS

Iteration and **Recursion** are the simple strategies for developing algorithms.

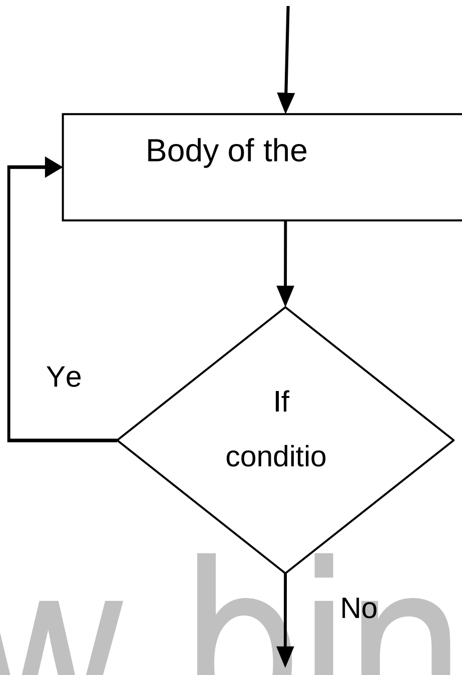
Iteration

Iteration is the process of repeating the same set of statements again and again until the condition becomes false. Iteration is done in two ways.

i) In iteration loop, the condition is checked at the beginning of the loop. If the condition is true, then the loop will be executed. If the condition is false, the loop will not be executed.



ii) Condition is checked at the bottom of the loop. If the condition is true, loop will be executed. Otherwise the loop will not be executed.



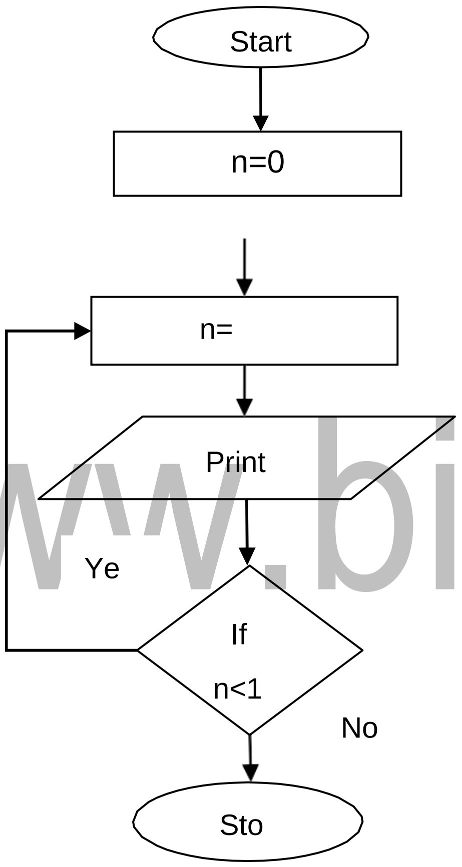
Example: Write an algorithm, pseudo code and draw the flowchart to print the numbers from 1 to 10.

Algorithm:

1. Start
2. Initialize n=0
3. Increment n by 1
4. Print n

5. Check condition n<10
    i. If yes goto step 3
    ii. Otherwise goto step 6
6. Stop

Flowchart:



Pseudo code:

```
BEGIN
OBTAIN n
INITIALIZE n=0
INCREMENT
n=n+1
DISPLAY n
IF n<10 REPEAT the loop
```

```
ENDIF
END
```

Recursion

Recursion is a programming technique that has a recursive function that calls itself again and again until the condition is reached.

Syntax:

```
function():
        function():
```

In the above syntax, function() is a function which call itself until the condition is reached.

Difference between recursion and iteration

Sl.No	Recursion	Iteration
1	Function calls itself until the base condition is reached.	Repetition of loop until condition false.
2	It keeps code short and simple.	It keeps the code longer.
3	It is slower due to overhead of maintaining the stack	It is faster.
4	It takes more memory.	It takes less memory.
5	Tracing is difficult if any problem occurs.	Tracing is easier if any problem occurs.

6.	It is a process always applied to a **function**	It is applied to the **set of instructions** which we want to get repeatedly executed .

Example: Calculating factorial of a number using recursion.

Algorithm:

1. Start
2. Read n
3. Call the function fact(n)
4. Print value of the factorial
5. Stop

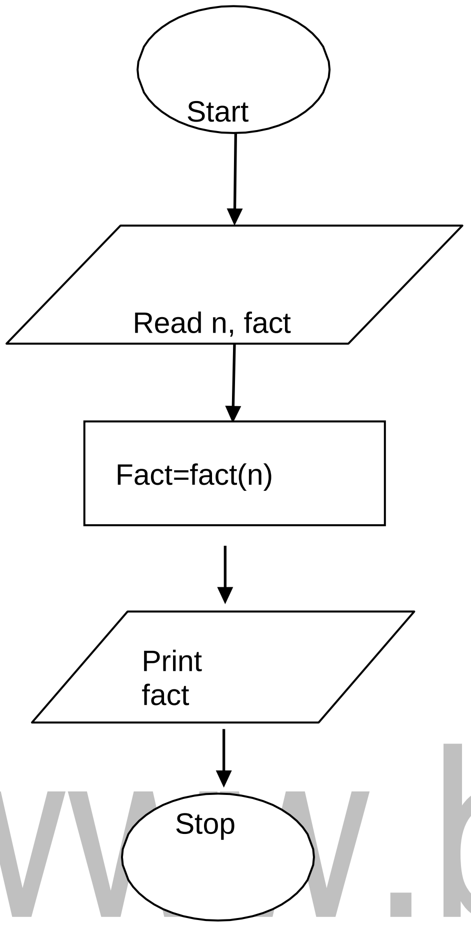
fact(n) function:

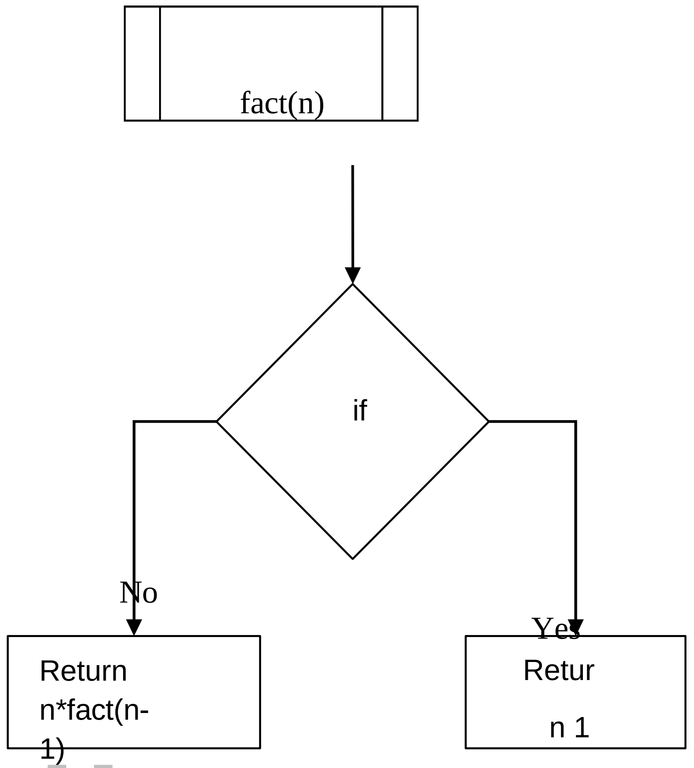
1. Start function
2. If n=0, return 1
3. If n>0 return function fact(n)
4. Return

Flowchart :

The recursive function used to calculate factorial of a number is

$$\text{fact}(n) = \begin{cases} \text{fact}(n) = 1 & \text{if } n=0 \\ n * \text{fact}(n-1) & \text{if } n>0. \end{cases}$$


Start
Read n, fact
Fact=fact(n)
Print fact
Stop


fact(n)
if
No
Yes
Return n*fact(n-1)
Retur n 1