GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING

UNIT – I

ALGORITHMIC PROBLEM SOLVING

ALGORITHM

- Algorithm is defined as step-by-step description of how to arrive at the solution of the given problem.
- It also defined as finite sequence of explicit instructions that when provided with a set of input values, produces an output and then terminates.
- An algorithm provides a blueprint to writing a program to solve a particular problem.
- It is an effective procedure for solving a problem in a finite number of steps.

Characteristics of Algorithm:

- Be precise
- Be unambiguous
- It has finite number of inputs
- It should be written in sequence
- It should conclude after a finite number of steps.

Qualities of Algorithm:

- Different algorithm may perform the same task with different set of instructions. Some algorithm is considered better than the other in solving the problem. The algorithm is written in such a way that it optimizes all necessary conditions.
- The factors that determine the quality of algorithm are:

➢ Accuracy: Algorithm should provide accurate result.
➢ Memory: It should take less memory.
➢ Time : It takes less time to execute
➢ Order : It must be in a sequential order.

Advantages:

* It is a step-wise description of a solution to a given problem, which makes it easy to Understand.

* It uses a definite procedure to solve a problem.

* It is independent on any programming language.

Disadvantages:

* It is not a computer program.

* It is time consuming. (An algorithm needs to be developed first, which is then converted into

Flowchart and then into a computer program.)

* It is difficult to show branching and looping statements.

Example1: Algorithm to find sum of two numbers.

Stpe1: Start

Step2: Read the values of a, b

Step3: Calculate c = a + b

Step4: Print the value of c.

Step5: Stop

Example 2: Algorithm to find area of rectangle

Step1: Start

Step2: Read length and breadth l, b

Step3: Calculate area of rectangle, area = l x b

Step4: Print the value of area.

Step5: Stop

BUILDING BLOCKS OF ALGORITHMS

Any algorithm can be constructed from four basic building blocks. These four building blocks are

* Statements
* State
* Control flow → Sequence, Selection, Iteration
* Function

Instructions / Statements:

* Instructions: Commands given to the computer that tell what it has to do.

* Statements: → A section (Segment) of code that represents a command.

→ Each and every line in an algorithm is called statement.

* Types of statement:

> Simple statement:

→ Assignment statement, return, goto etc

> Compound statement:

→ For, while, if, if-else etc.

State:

* Each and every action in an algorithm is called state.

* An algorithm is a finite number of steps for accomplishing a goal which, given an initial state, will terminate in a defined end state.

Control flow:

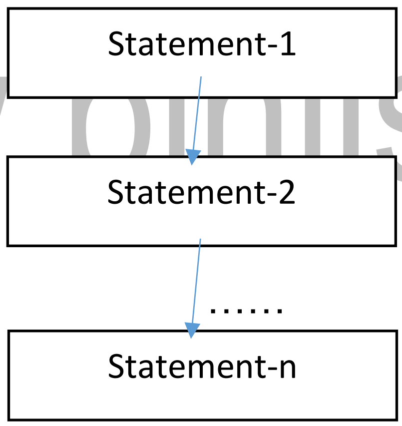
* It is also called flow of control. It is the order in which instructions or statements are executed. There are three types of control flow, they are

- Sequential control structure → Action
- Selection control structure → Decision (or) Branching
- Iteration control structure → $Repetition\ (or) Looping$

* Sequential control structure (or) sequence structure:

→ It is the most common form of control structure.

→ Each statement in the program is executed one after another in the given order.



→ Example:

Add two numbers

Step1: Start

Step2: Read values of a and b

Step3: Add the values of a and b

c = a + b

Step4: Print the value of c

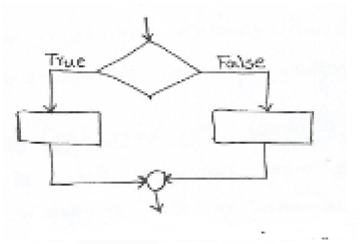
Step5: Stop.

- **Selection control structure:**

→ It is the representation of a condition and choice between two actions.

→ The choice made depends on whether the condition is true or false. It is also called a Decision.

→ This structure is sometimes referred as if-then-else structure because it directs the program to perform in this way: If condition A is true then perform Action X else perform Action Y.



→ Example: Algorithm to check the given number is odd or even.

Step1: Start

Step2: Read the value of num

Step3: Check if num % 2 = = 0 then

Step3.1: Display (Print) the given number is even

Step3.2: Else print the given number is even

Step4: Stop

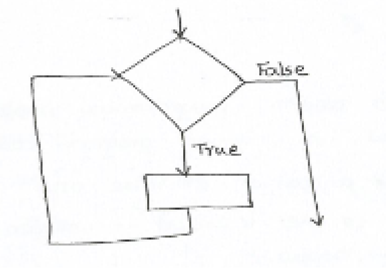
* Iteration control structure:

→ This process is also called as repetition or looping.

→ It is the process of repeating a set of instructions.

→ There are two types of iteration:

- Count controlled Iteration : It will repeat a set of instructions for a specific number Of times.
- Condition controlled iteration : It will repeat the instructions until a specific Condition is met.



→ Example: Algorithm to find factorial of a number

Step1: Start

Step2: Initialize the variable,

fact = 1 and i =1

Step3: Read the value of num

Step4: Repeat the following steps until i = num

fact = fact * i

i = i + 1

Step5: Print the value of fact

Step6: Stop.

Function:

* A function is a set of instructions that are used to perform a specified task which repeatedly occurs in the main program.

* There are two types of function

> Built – in function

> User defined function

* The built – in functions are predefined set of functions.

* The user defined functions are defined by the user, according to the user requirements.

NOTATION;

PSEUDOCODE:

* Pseudo code is another programming analysis tool, which is used for planning program code.

* Pseudo means imitation or false and code refers to the instructions written in programming Language.

* It is also called program Design Language (PDL).

* Pseudo code is somewhat halfway in between English and a programming language.

* Here pseudo code is detailed yet readable and it ensures that actual programming is likely to match the design specification.

* Pseudo code keywords:

> Input: READ, OBTAIN, GET and PROMPT

> Output: PRINT, DISPLAY and SHOW

> Compute: COMPUTE, CALCULATE and DETERMINE

> Initialize: SET and INITIALIZE

> Add one: INCREMENT

> Sub one: DECREMENT

* Pseudo code Guidelines:

→ Statements should be written in simple English.

→ It should be programming language independent.

→ Steps must be understandable.

→ Pseudocode must be concise

→ Keywords must be capitalized.

→ Each instruction should be written in a separate line.

→ Each statement should express just one action for the compiler.

→ Each set of instructions are written from top to bottom with only one entry and one exit.

Advantages:

* It is easy to develop a program from a pseudo code than with a flowchart.

* It is easily modified

* It is compact

* It is language independent

Disadvantages:

* It doesn't provide visual representation of the program lagic.

* There is no accepted standards for writing pseudo code.

* There is no real formatting or syntax rules.

* It is not used to understand the flow of the program control.

Example: (i) Pseudo code for finding sum of two numbers


```
BEGIN
        READ the values of a, b
        CALCULATE the sum, c = a + b
        PRINT the value of c
END
```

(ii) Pseudo code for finding area of circle with radius 'r'

```
BEGIN
        READ the value of r
        CALCULATE the area of circle
            area = 3.14 * r * r
        PRINT the value of area
END
```

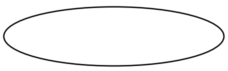
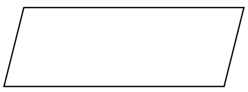
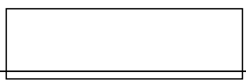
Flowchart:

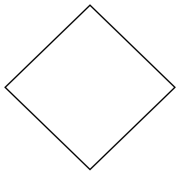
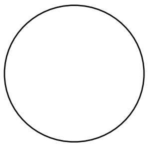
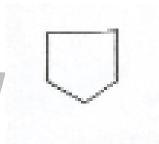
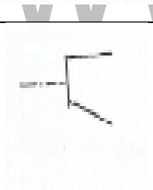
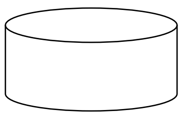
* Flowchart is the diagrammatic representation of an algorithm, in which the steps are Represented in the form of different shapes of boxes and logical flow is indicated by Interconnecting arrows.

* The boxes represent operations and the arrows represent the data flows (the sequence in Which the operations are implemented.

* The purpose of the flowchart is to help the programmer in understanding the logic of the Program.

* <u>Flowchart symbols:</u>

Symbol	Symbol Name	Description
	Flow lines	Used to connect symbols. It indicates the sequence of steps and the direction of flow of control.
	Terminal	Used to represent the start or stop (end) of the program.
	Input/output	Used to represent the input or output statement, in the program.
	Processing	

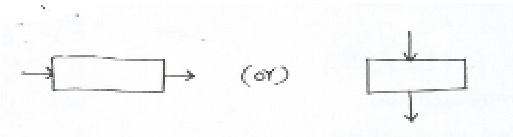
		Used to represent the processing function of a program.
	Decision	Used to denote a decision to be made.
	Connector	Used to join different flow lines.
	Off-page connector	Used to indicate that the flowchart continues on the next page.
	Annotation	Used to provide additional information about another flowchart symbol.
	Magnetic Disk	Used to represent data input or output and to a magnetic disk.

* <u>Guidelines for drawing flowchart:</u>

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

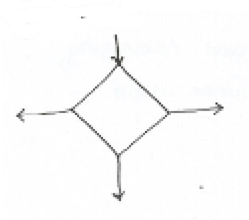
> The flowchart must have a logical start and stop.

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

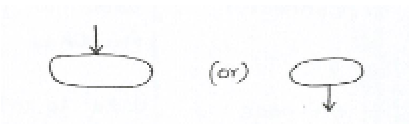


> Only one flow line should enter a decision symbol, but two or three flow lines should leave the

decision symbol.

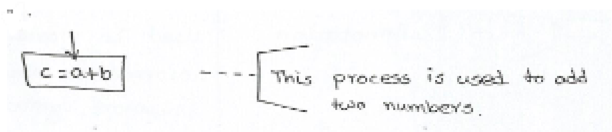


> Only one flow line is used with a terminal symbol.

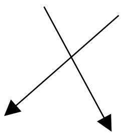


> Within standard symbols write briefly. If necessary use annotation to describe data or

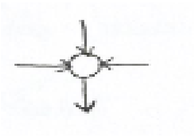
Process more clearly.



> Intersection of flow lines should be avoided.



> Connector symbol is used to reduce the number of flow lines.



> It is useful to list the validity of the flowchart with normal test data.

* <u>Benefits of flowcharts:</u>

> Communication

> Effective Analysis

> Makes logic clear

> Useful in coding

> Proper testing and debugging

> Appropriate documentation

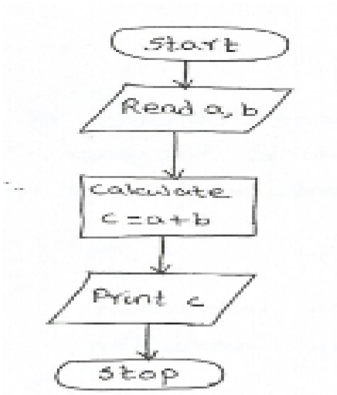
* <u>Limitations of flowcharts:</u>

> Complex

> Difficult to modify

> No update

* <u>Example:</u> Flowchart to find sum of two numbers.



<u>PROGRAMMING LANGUAGE;</u>

* A computer is the ideal machine to execute computational algorithms.

* A computer can perform arithmetic operations.

* It can also perform operations with conditional/ branching instruction.

* Types of Languages used in computer programming:

> Low level (or) Machine Language

> Assembly Level Language

> High level (or) programming Language.

* Machine Language:

> It consists of binary numbers that encode instructions for the computer.

> Every computer has its own machine language.

> Example: 101011101

100110110

* Assembly Language:

> An assembly language consists of mnemonics

> There is one mnemonic for each machine instruction.

> Each assembler instruction is a mnemonic

> Example:

Start

add x, y → one instruction

Sub x, y → one instruction

End

- High level programming Language:

- It consists of English like languages.
- It can be easily translated into machine instruction.
- The sentences written in high level languages are called "statements".
- Example: python program

```
x = 10
y = 20
z = x + y
Print "The sum is", z
```

Output:

The sum is 30

- Types of High Level programming Languages

Computer programming languages are used to communicate instructions to a computer. They are based on certain syntactic and semantic rules, which define the meaning of each of the programming language constructs. They are divided into the following categories:

- Interpreted programming Languages
- Functional programming Languages
- Compiled programming Languages
- Procedural programming Languages
- Scripting programming Languages
- Mark-up programming Languages
- Logic-Based programming Languages
- Concurrent programming Languages
- Object-Oriented programming Languages

- Interpreted programming Languages:

→ It is a programming language for which most of its implementations execute Instructions directly, without previously compiling a program into machine - language Instructions.

→ Example: BASIC, Pascal, Python

- Functional programming Languages:

→ It define every computation as a mathematical evaluation (calculation). They focus On the application of functions.

→ Example: F, Haskell, Q

- Compiled programming Languages:

  → It is a programming language whose implementations are typically compilers, and not interpreters.

  → Example: C, C++, Java

- Procedural Programming Languages:

  → Also called imperative programming languages.

  → A procedure is a group of statements that can be referenced through a procedure Call.

  → It help in the reuse of code.

  → Example: Hyper Talk, MATLAB.

- Scripting Programming Languages:

  → It is a programming language that control an application

  → Scripts can execute independentof any other application.

  → Example: PHP, Apple script, VBScript.

- Mark-up programming Languages:

  → It is an artificial language that uses annotations to text that define how the text is to Be displayed.

  → Example: HTML, XML, XHTML

- Logic – Based programming Languages:

  → It is a type of programming paradigm which is largely based on formal logic.

  → It is a set of sentences in logical form, expressing facts and rules about some Problem domain.

  → Example: ALF, Leda, Prolog.

- Concurrent Programming Languages:

  → It is a computer programming technique that provides for the execution of Operations concurrently, either within a single computer, or across a number Of systems.

  → Example: ABCL, E, Limbo

- Object – Oriented programming Languages:

  → It is a programming paradigm based on the concept of “objects”, which may contain data, in the form of fields, often known as attributes: and code, in the form of Procedures, often. Known as methods.

→ Example: Agora, Beta, Lava

ALGORITHMIC PROBLEM SOLVING:

- Algorithms are procedural solutions to problems.
- These solutions are not answers but specific instructions for getting answer.

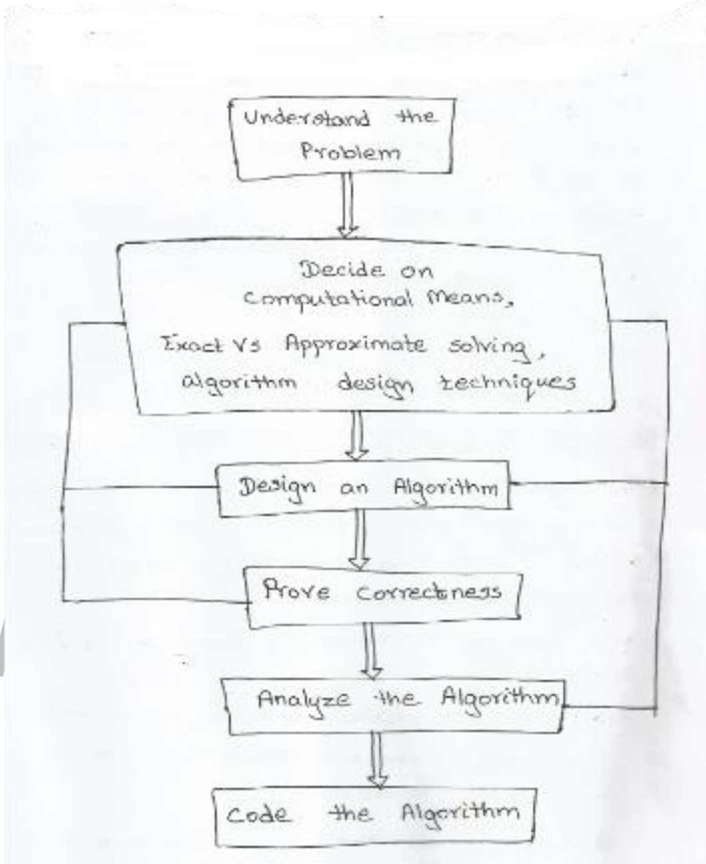


Fig.: Steps in algorithmic problem solving

- Understanding the problem:
- Before designing an algorithm we need to understand completely the given problem.
- Read the problem's description carefully and ask questions about the problem.
- An input to an algorithm specifies an instances of the problem that the algorithm solves.
- Ascertaining the capabilities of the computational Device:
- Once we completely understand the problem, we need to ascertain the capabilities of the computational device.
- Sequential Algorithms:

→ The instructions are executed one after another i.e., one operation at a time.

- Parallel Algorithms:

→ The instructions are executed simultaneously

i.e., multiple operations at a time

- Choosing between Exact and Approximate problem solving:
- The next step is to choose how to solve the problem.
- Solving the problem exactly is called an exact algorithm.
- Solving a problem approximately is called an approximation algorithm.
- We select an approximation algorithm for three reasons:
  - (i) There are problems that cannot be solved exactly.

    Example: extracting square roots, solving non-line equations.

    (ii) Available algorithms are slow because of its complexity.

    (iii) An approximation algorithm can be part of sophisticated algorithm that

    Solves a problem exactly.

- Deciding on appropriate Data structure:
- Data structure is a scheme of organizing related data items.
- Algorithms + Data structure = programs
- Algorithm Design Techniques:
- It is a general approach to solve problem algorithmically
- It provides guidance for designing new algorithms.
- It is used to classify algorithms according to design area.
- Methods of specifying an algorithm:
  - Step by step form
  - Pseudo code
- Proving an Algorithm's Correctness:
- Once an algorithm is specified it has to be proved for its correctness.
- A technique for proving correctness is to use mathematical induction.
- Because an algorithm's iterations provide a natural sequence of steps needed for such proofs.
- But in order to show that an algorithm is incorrect, we need just one instance of its input for which the algorithm fails.
- Analysing an algorithm;

- Characteristics used for analysing an algorithm are

(i) Efficiency:

- ➢ Time efficiency: Denotes how fast the algorithm runs.
- ➢ Space efficiency: Indicates how much extra memory it occupies.

(ii) Simplicity:

- ➢ Simpler algorithms are easier to understand and to program.

(iii) Generality:

- ➢ There are two issues:

(i) Generality of the problem the algorithm solves.

(ii) Set of inputs it accepts.

- Coding an Algorithm:
- Algorithms are implemented as programs.
- Unless the correctness of a algorithm is proven, the program can't be considered correct.
- The validity of programs is checked by testing.

SIMPLE STRATEGIES FOR DEVELOPING ALGORITHMS:

- Algorithms are used to manipulate the data for a given problem.
- For a complex problem, its algorithm is often divided into smaller units called modules.
- This process of dividing algorithm into modules is called modularization.
- The advantage of modularization is :
  → It makes the complex algorithm simpler to design and implement.
- Each module can be designed independently.
- There are two main approaches to design an algorithm
  - ➢ Top – down approach
  - ➢ Bottom – up approach
- Top – down Approach:
- It starts by dividing the complex algorithm into one or more modules.

- These modules can be further divided into one or more sub – modules.
- This process is iterated until the design level of module complexity is achieved.
- <u>Bottom – up Approach:</u>
- It is the reverse of top-down approach.
- It starts designing the most basic or concrete modules and then proceed towards designing higher level modules.
- In this approach sub –modules are grouped together to form higher level module.
- This process is repeated until the design of the complete algorithm is obtained.
- <u>Iteration:</u>
- Iteration means, executing one or more steps for a number of times.
- It can be implemented using while and for loop.
- These loops execute one or more steps until some condition is true.
- <u>Example:</u> Algorithm to print 'N' numbers

  Step1: Start

  Step2: Read the value of N

  Step3: Set the value of i = 1

  Step4: Repeat step 5 and 6 until i< = N

  Step5: Print the value of i

  Step6: Increment, i = i + 1

  Step7: Stop
- <u>Recursion:</u>
- Recursion is the process of calling the same function itself again until some condition is satisfied.
- <u>Example:</u> Algorithm to find factorial of a number.

  Step1: Start

  Step2: Read the value of N

  Step3: Call function, factorial (N)

  Step4: Stop

<u>User Defined function:</u> factorial (N)

Step1: Initialize f = 1

Step2: Check if N = = 1 then return 1

Step3: Else, f = n * factorial (n-1)

Step4: Print the value of f

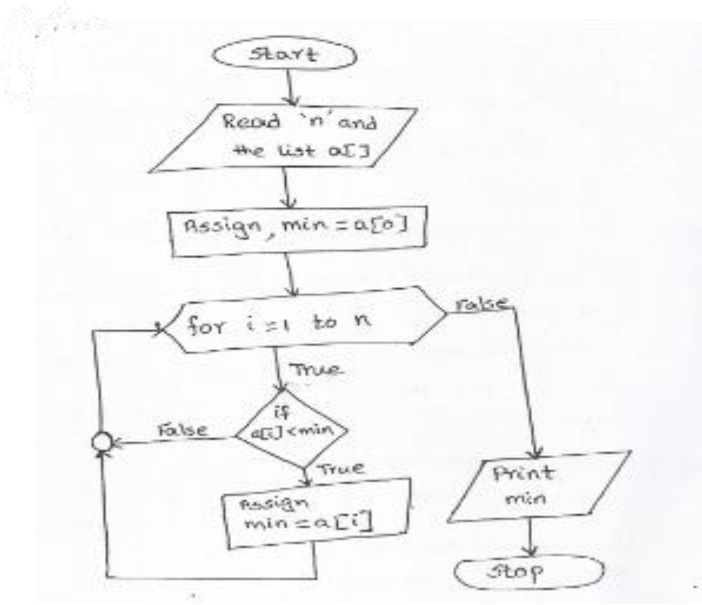
ILLUSTRATIVE PROBLEMS

1. Find Minimum in a list

- Problem Description:
- 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.
- Assume the first element as the minimum element and start comparing with the next (second) element.
- If the next element is smaller than assume that element as the minimum, and keep repeating the process till the last element.
- Finally obtain the minimum element.

ALGORITHM	PSEUDOCODE
Step1 : Start Step2 : Read the limit of list as n Step3: Read the 'n' elements of list a (). Step4 : Assign, min = a[0] Step5 : For i = 1 to n repeat Step 5.1 and 5.2 Step5.1 : Check if a[i] < min Step5.2 : Then assign, min = a[i] Step6 : Print the value of min Step7 : Stop	BEGIN READ the limit of list as 'r' READ the 'n' elements of list a() ASSIGN min = a[0] FOR i = 1 to n REPEAT CHECK IF a[i] < min THEN min = a[i] PRINT the value of Min END

Flowchart:



2. Insert a card in a list of sorted cards

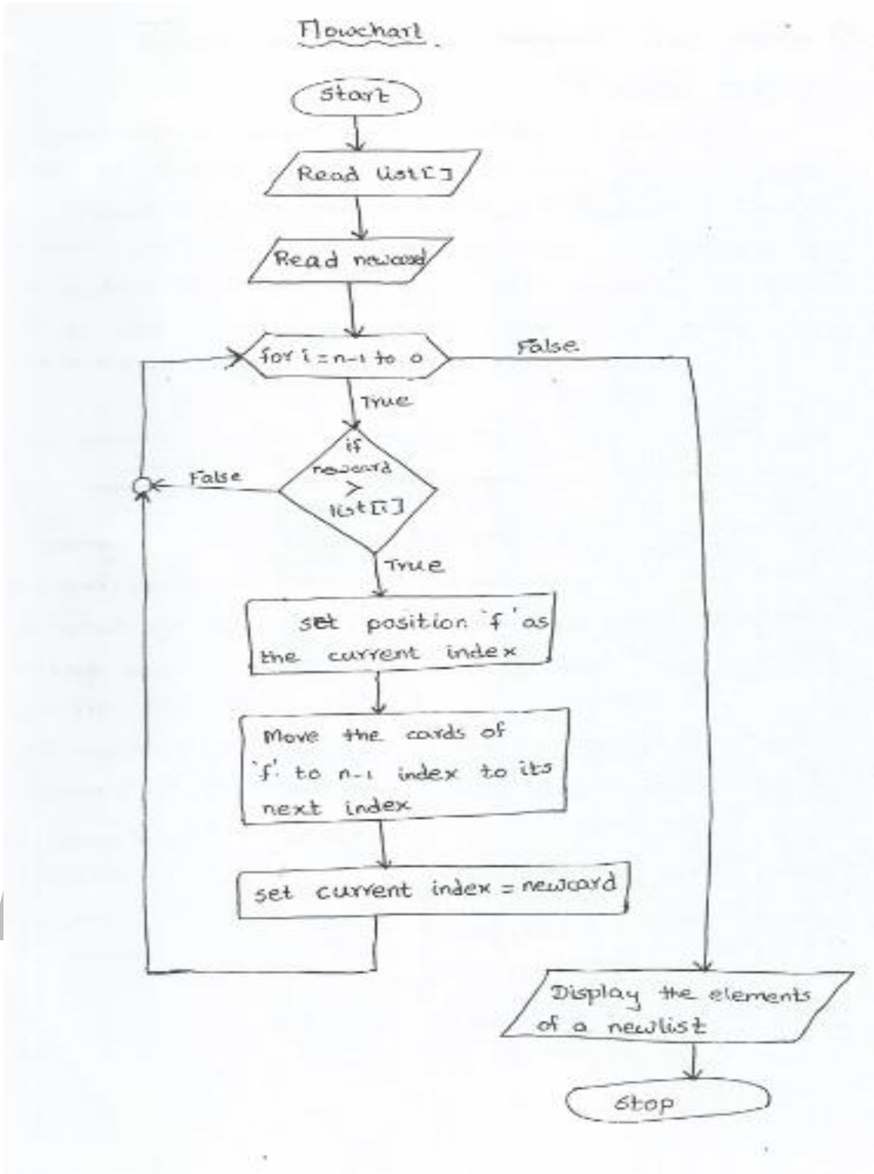
Problem Description:

Playing cards is one of the techniques of sorting. To insert a card in a list of sorted cards the steps are as follows:

- Insert a new card in the appropriate position by comparing each element's value with the new card.
- When the position is found, move the remaining element (elements from that position) by one position) by one position up.
- The new card is inserted at the current position.

Algorithm	Pseudo code
Step1 : Start Step2 : Read the elements of a list in sorted Order, list() Step3 : Read a new element (card) to be Inserted. Step4 : Traverse the list For i = n-1 to 0 Step4.1 : Check if new card > list [*i*] then	BEGIN READ the elements of a List, list() in sorted order READ a new element, (new card) to be inserted Traverse the list FOR i = n-1 to 0 REPEAT CHECK IF new card> list [*i*] THEN

Step4.2: Set position 'f' as the current index. Step4.3 : Move the cards of f^{th} Index to n-1 index to its next index. Step5: Set new card to the current index. Step6 : Display all the card elements of a new list. Step7: Stop.	SET position 'f' as the Current index Move the cards of f^{th} Index to n-1 index To its next index SET new card to the Current index. DISPLAY all the card Elements of a new list END.

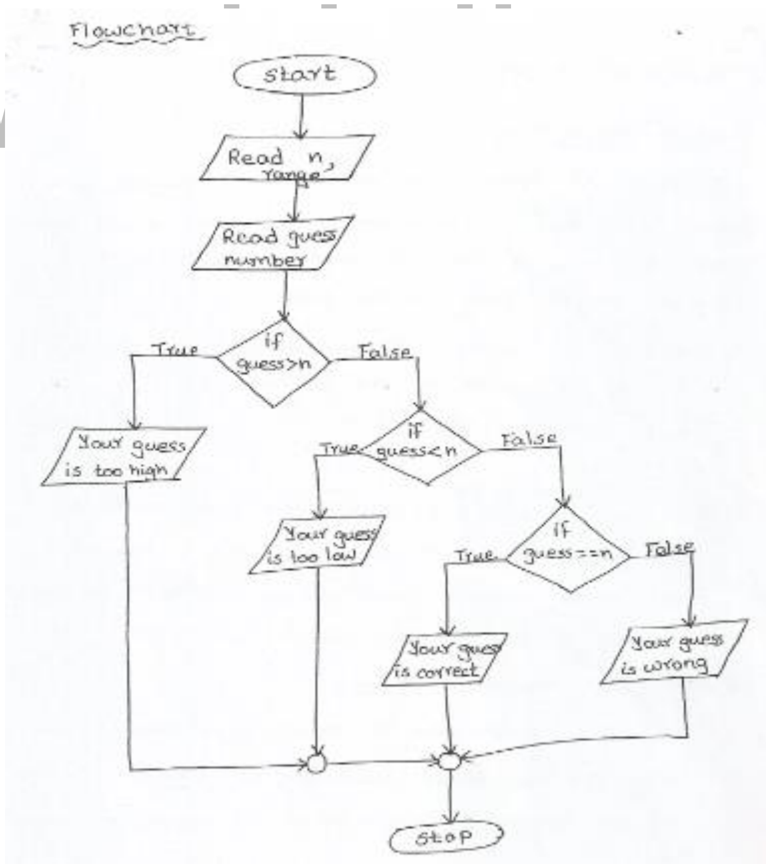


3. Guess an Integer Number in a Range

Problem Description

To guess a number in a range of elements, the input guessing number should be restricted to the specified range. The random number is automatically generated by the system and it can be stored in a variable. If both the guess number and random number are same, it should print "Good Jb" else it must print whether it is greater are lesser than the random number.

ALGORITHM	PSEUDOCODE
Step1 : Start	BEGIN
Step2: Read a number r 'n' range.	READ a number 'n', range
Step3 : Read an guess number	READ an guess number
Step4 : If (guess>n) then	IF (guess > n) THEN
Step4.1: Print "your guess too high".	PRINT "your guess too high".
Step5: Elif (guess <n) then	ELIF (guess < n) THEN
Step5.1: Print "your guess too low".	PRINT "your guess too low".
Step6 : Elif (guess = = n) then	ELIF (guess = = n) THEN
Step6.1: Print "your guess is correct".	PRINT "your guess is correct".
Step7 : Else print "your guess is wrong"	ELSE PRINT "your guess is wrong".
Step8 : Stop	END

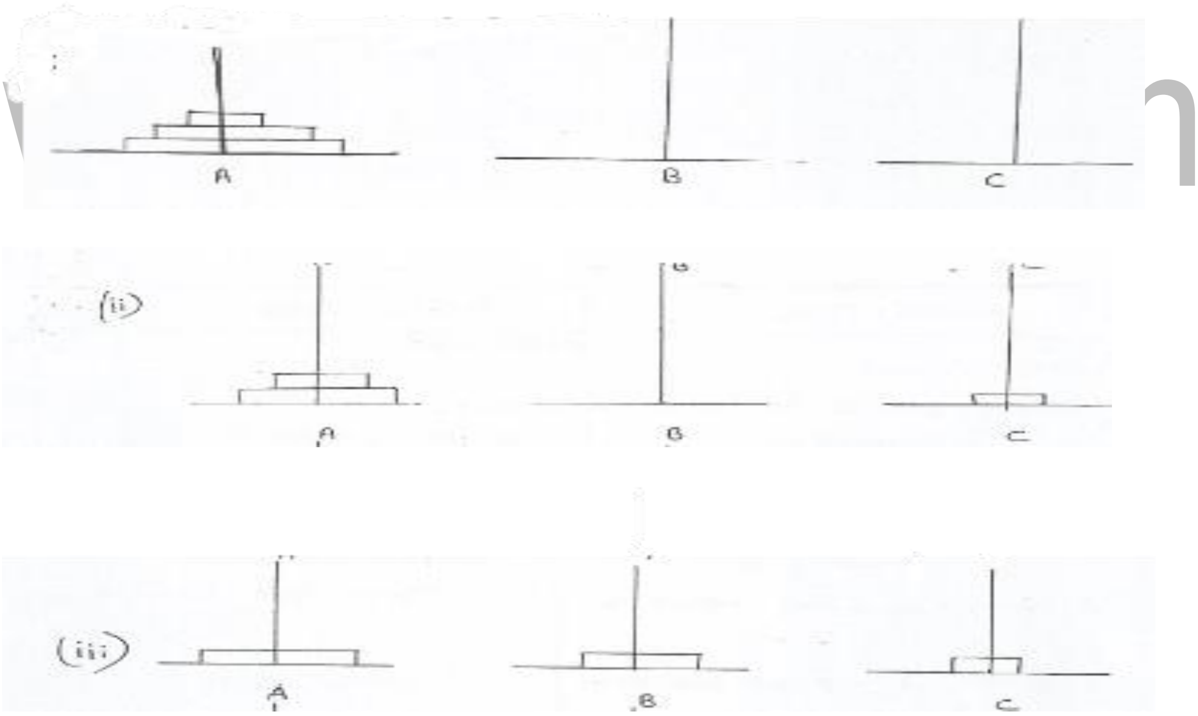


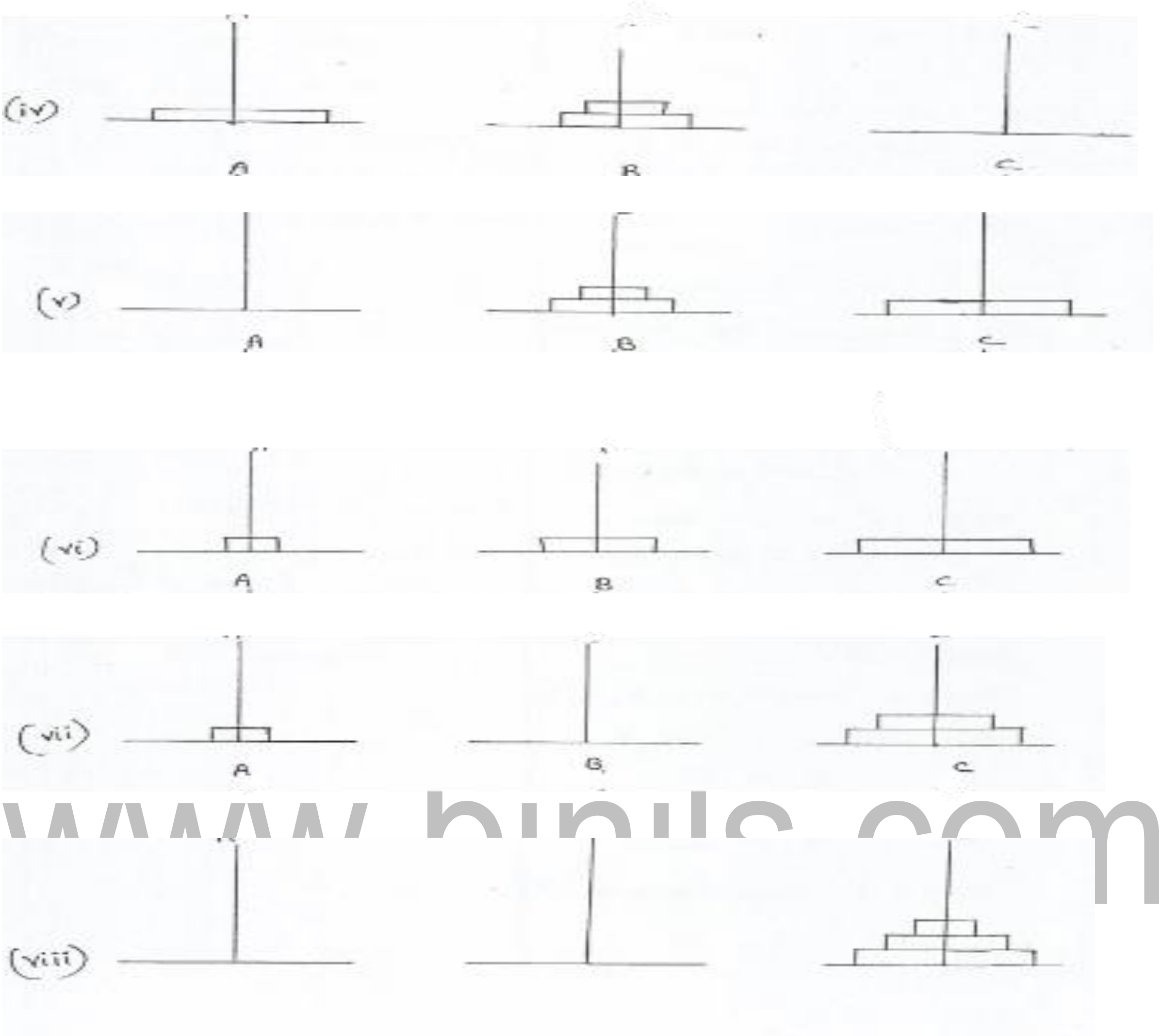
4. Towers of Hanoi

Problem Description:

- Tower of Hanoi is a mathematical puzzle with three rods and 'n' number of different sized disks, each disk has a hole in centers, allowing it to be stacked around any of the poles.
- Initially the disks are stacked on the left most pole in the order of decreasing size, i.e., the largest disk at the bottom and the smallest on the top.
- The aim (objective) of this game is to move the disks from the left mast pole (rod) to the right most poles, without ever placing a larger disk on the top of the smaller disk.
- Rules for Towers of Hanoi:
    - ➢ Only one disk may be moved at a time.
    - ➢ Only the top most disk can be moved
    - ➢ Only the smaller disk can be placed above the larger disk.

Solution: Let us consider n =3 and three rods (pegs) named A, B and C.





ALGORITHM	PSEUDOCODE
Step1 : Start	PROCEDURE
Step2 : Define the function tower()	TOWER (N, A, C, B)
Step3: Read the number of disk N.	BEGIN
Step4 : Initialize the pegs	IF N = = 1 THEN
A = Source, B = aux,	Move disk from A to C
C = dest.	ELSE
Step5 : Call the function	TOWER (N-1, A, B,C)
tower (N, A, C, B)	move disk from A to C
Step6 : Stop	TOWER (N-1, B,C,A)
User Defined Function:	ENDIF

Step1 : Define the function tower (N, A, C, B) Step2 : if N = = 1 then move disk from A to Step3 : Else Step3.1 : tower (N-1, A, B, C) Step3.2 : move disk from A to C Step3.3 : tower (N-1, B, C, A)	END END PROCEDURE // N → no. of disks. 3 pegs → A, B, C Source aun Dest