UNIT- I
ALGORITHMIC PROBLEM SOLVING

Algorithms, Building blocks of algorithms (statement, 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.

Computer *(2 marks)*

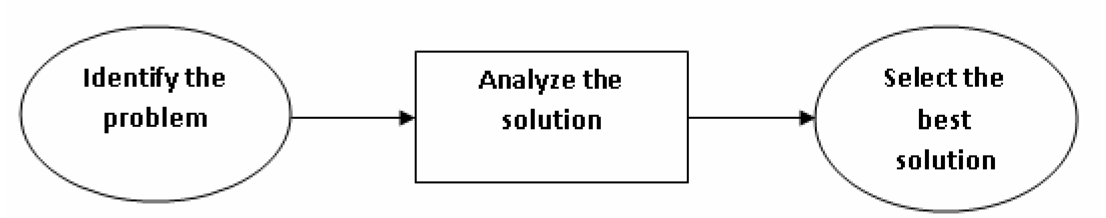
- Electronic machine that,
    - Accepts data from the environment
    - Processes the data by performing calculations and operations
    - Generates output result to the environment

Program *(2 marks)*

- A program is a **step by step series of instruction** given to the computer to produce a desired output.
- An **instruction** is an basic commands

Problem solving *(2 marks)*

- Problem solving is the sequential process of analyzing information related to given situation and generating appropriate response options

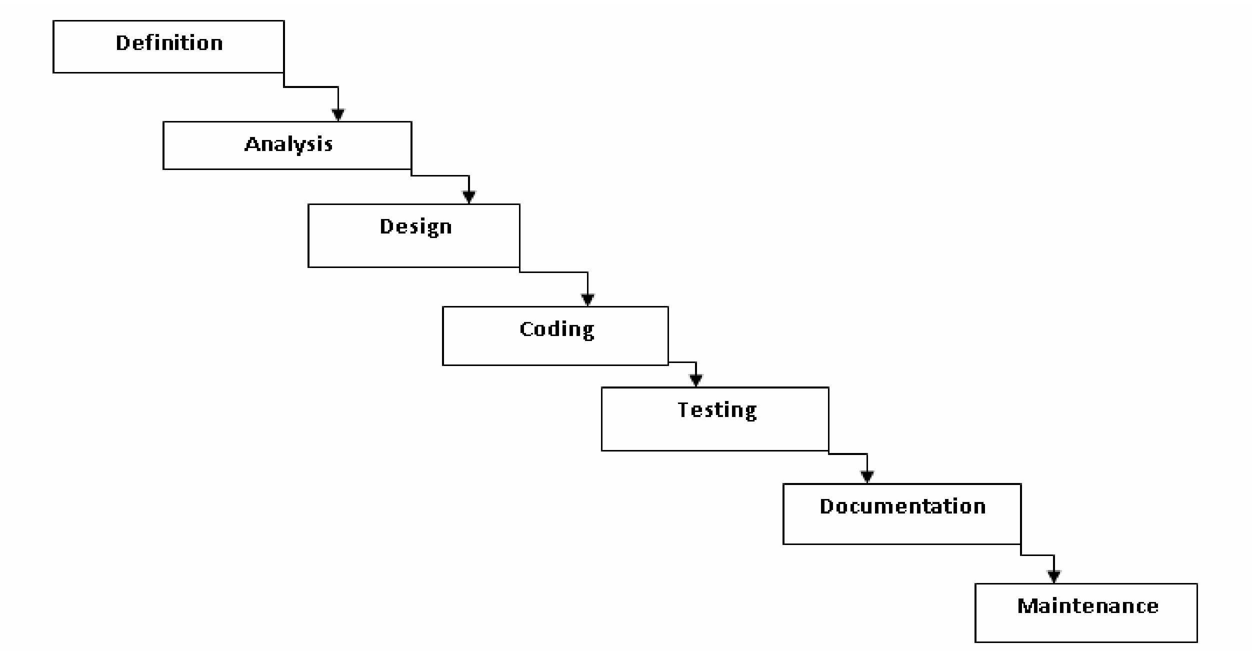


Problem solving Logic

- Steps involves in Problem solving,
    - Programmer understands the problem.
    - Analyze the problem to find out different way to solve it
    - Determines multiple solutions to the problem.
    - Decides and selects a single exact solution
    - Selected solution is represented in a detailed step-by-step manner
    - It is coded using suitable programming language
    - Execute the program.

<u>Problem solving Methodology</u> (Software Life cycle) *(8 marks)*

- There are seven steps in problem solving Methodology



Problem solving Methodology

1. Problem Definition

- Understand the description of the problem to solve
- **Example:**

  To find average of 5 numbers

  Asking questions like,
  - What input data is available?
  - Is anything missing?
  - What output data I trying to produce?
  - What I am going to compute?

2. Problem Analysis

- Original problem is analyzed and divided into many sub problems
- sub problems are easier to solve
- each sub problems is divided into further smaller ones
- this fragmentation continued to achieve simple solution

3. Designing the Problem

- Designing is a process used to develop a program

- It requires 3 steps, they are
    1) **Algorithm:** step-by-step description of the solution.
    2) **Flowchart:** Diagrammatic representation of the solution.
    3) **Pseudo code:** It is written for selected solution. It gives logic of the program

4. Coding *(2 marks)*

- Algorithms can't executed directly by the computer, it has to be translated into a programming languages.
- The process of translating the algorithms into programs using programming languages is known as ***coding***.

5. Program testing and debugging

- Testing means running the program.
- Check the output results with different inputs.
- If the output is incorrect, modify the program to get correct results.
- ***Debugging*** is the process of finding and correcting the errors in the program.

6. Documentation

- It includes,
    - problem definition
    - Design documents
    - History of program development
    - User manual

7. Program Maintenance

- Finding and eliminating previously undetected program errors.
- Modify the current program, often to improve its performance.
- Adding new features to the program
- Updating the documentation

ALGORITHM *(16 marks & 8 marks)*

Algorithm *(2 marks)*

"It is an ordered sequence of finite, well defined, unambiguous instruction for completing a task"

- It is a step-by-step procedure for solving a task.
- English like representation of the logic.
- Steps must be,
    - Ordered
    - Unambiguous
    - Finite in number

- For a particular task, different algorithms can be written.
- Different algorithms differ in their requirements of time and space.

Example: To find the greatest among 3 numbers. ***(8 marks)***

Algorithm

Steps:

1. Start
2. Read 3 numbers A, B, C.
3. Compare A & B. If A is greater, perform step 4 else perform step 5.
4. Compare A & C. If A is greater, output "A is greatest"
5. Else output "C is greatest".
6. Perform step 6.
7. Compare B & C. If B is greater, output "B is greatest"
8. Else output "C is greatest".
9. Stop

Characteristics of Algorithm *(2 marks)*

- Simple and easy to understand
- Should be clear and unambiguous
- It look like normal English
- Includes finite sequence of steps in order
- Instruction should not be repeated infinitely
- Should covers logic of the problem
- Desired result should be obtained at the end of the algorithm

Qualities of Algorithm *(2 marks)*

The following are the primary factor used to judge the quality of the algorithm

1. **Time**
   - To execute programs, computer takes some amount of time
   - The lesser is the time required ,the better is the algorithm
2. **Memory**
   - To execute programs, computer takes some amount of memory storage
   - The lesser is the memory required ,the better is the algorithm
3. **Accuracy**
   - Multiple algorithm may provide suitable solution to the problem
   - Some of these may provide accurate results than other algorithm
   - Such algorithm are suitable
4. **Sequence**
   - Instruction in the algorithm must be in order
   - Some instruction of an algorithm may be repeated number of times until a particular condition satisfy.

5. **Generability**
    - The designed algorithm must solve a single isolated problem.
    - So algorithms must be generalized.

<u>Advantage of Algorithm</u> *(2 marks)*

- Simple to understand
- Step wise description of the solution
- Easy to debug
- Independent of programming languages
- Using algorithm problem can be broken down into smaller pieces or steps
- Each step of the algorithm can be easily coded into its equivalent high level language

<u>Disadvantage of Algorithm</u> *(2 marks)*

- Time consuming process
- Difficult to show branching and looping statement
- Algorithm is not computer program, rather a concept of how a program should be written

BUILDING BLOCKS OF AN ALGORITHM *(16 marks & 8 marks)*

<u>Building Blocks of an Algorithm</u>

- Algorithm can be constructed from basic building blocks
- The ***building blocks of algorithms*** are ***(2 marks)***
    1. Instruction/statements
    2. State
    3. Control flow
        i) Sequence
        ii) Selection (conditional)
        iii) Iteration(looping)
    4. Function

1. Instruction/statements

- An effective procedure for solving a problem in a finite number of steps
- Instruction must be in order
- Time taken to execute all instruction of the algorithm should be finite and within a reasonable limit.

2. State

- It specify the state or condition for program termination

- It also specify the position in the algorithm

3. **Control Flow *(2 marks)***
    - It is an order in which individual statements, instruction are evaluated
    - Control flow allow the program to make choice, change direction or repeat action
    - The basic control flow needed for writing good and efficient algorithm are
        - Sequence
        - Selection (conditional)
        - Iteration(looping)

i) Sequence *(2 marks)*

- In sequence control flow,
    - Instructions are executed in linear order
    - One after the other

Example: Algorithm to find sum of two numbers ***(2 marks)***

```
Step 1: Start
Step 2: Read two numbers A and B
Step 3: Calculate sum= A + B
Step 4: Print the sum value
Step 5: Stop
```

ii) Selection (conditional) *(2 marks)*

- In selection control flow,
    - Instructions are executed based on condition
    - If condition is true ,one path is followed
    - If condition is false, another path is followed

Example: Algorithm to find biggest of two numbers ***(2 marks)***

```
Step 1: Start
Step 2: Read two numbers A and B
Step 3: if (A>B) then
                Print 'A' is bigger
        else
                Print 'B' is bigger
Step 4: Stop
```

iii) Iteration (looping) *(2 marks)*

- In iterative control flow, a same set of instruction executed repeatedly until it satisfy particular condition

Example: Algorithm to compute and print average of 10 numbers ***(2 marks)***

```
Step 1: Start
Step 2: total=0, average=0
Step 3: for 1 to 10
Step 4: Read number
Step 5: total=total + number
Step 6: END for
Step 7: average=total/10
Step 8: Print average
Step 9: Stop
```

4. **Function** ***(2 marks)***
   - For complex problems, the task is divided into number of smaller sub-task during algorithm design
   - A function is a block of organized, reusable code that is used to perform single, related actions.
   - Provide better modularity for application
   - Provide high degree of code reuse

Example: Algorithm to find addition of two numbers

```
Step 1: Start
Step 2: Read two numbers 'a' & 'b'
Step 3: Call addition (a, b)
Step 4: Print the sum value
Step 5: Stop

addition (a, b)

Step 1: Sum= a + b
Step 2: return sum
```

PSEUDOCODE *(16 marks & 8 marks)*

<u>Pseudocode</u> *(2 marks)*

"Pseudocode is an outline of a program written in a form that can be easily converted into real programming statement"

- Pseudo means 'false or imitation'
- Code means 'instruction'
- Pseudocode is
  - short
  - Readable

- Formally styled English languages used for explaining an algorithm

- Short hand way of describing a computer program
- It is not based on any programming languages
- It gives structure of the program before the actual coding
- It is also called ***program design language***

Keywords used in Pseudocode *(2 marks)*

- Pseudocode uses some keywords to denote programming processes, they are
  - **Input:** INPUT, GET, READ, OBTAIN
  - **Output:** PRINT, DISPLAY, SHOW, OUTPUT
  - **Compute:** COMPUTE, CALCULATE, DETERMINE
  - **Initialization:** SET, INITIALIZE
  - **Add one:** INCREMENT

Example: Pseudocode to find addition of three numbers. ***(2 marks)***

```
READ A, B, C
COMPUTE the sum by adding A, B, C
DISPLAY the sum
```

Rules for writing Pseudocode *(2 marks)*

Or

Guidelines for preparing Pseudocode *(2 marks)*

- Statements should be written in English and programming language independent
- Write one statement per line
- Capitalize initial keywords
- Each set of instruction must be written from top to bottom
- End multiline structure
- Should describe logical plan to develop a program

Control structure used in Pseudocode *(16 marks & 8 marks)*

- There are three control structure used in pseudocode, they are

1. **Sequence *(2 marks)***
   - In sequence control flow,
     - Instructions are executed in linear order
     - One after the other

   Example: Pseudocode to find sum of two numbers ***(2 marks)***

```
START
READ the values of A and B
```

```
CALCULATE sum by adding A with B
PRINT the sum value
STOP
```

2. **Selection (conditional)** ***(2 marks)***
   - In selection control flow,
     - Instructions are executed based on condition
     - If condition is true ,one path is followed
     - If condition is false, another path is followed

Example: Pseudocode to find biggest of two numbers ***(2 marks)***

```
START
READ the values of A and B
IF (A>B) THEN
            PRINT 'A' is bigger
       ELSE
            PRINT 'B' is bigger
STOP
```

3. **Iteration (looping)** ***(2 marks)***
   - In iterative control flow, a same set of instruction executed repeatedly until it satisfy particular condition

Example: Pseudocode to compute and print average of 10 numbers ***(2 marks)***

```
START
INITIALIZE total=0, average=0
FOR (1 to 10)
READ number
COMPUTE total =total + number
END FOR
COMPUTE average=total/10
PRINT average
STOP
```

<u>Advantage of Pseudocode</u> ***(2 marks)***

- It can be written easily
- It is compact and easy to modify
- It can be readable
- Easy to understand the general working of the program.
- Gives the sketch of structure of program.
- For writing pseudo code, programmer need not know the programming language.

Disadvantage of Pseudocode *(2 marks)*

- It is not visual
- Not get the picture of design
- No standard rules for writing pseudocode
- Not used to understand the flow of program control

FLOW CHART *(16 marks & 8 marks)*

Flow Chart *(2 marks)*

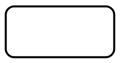
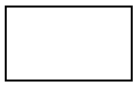
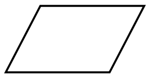
- Diagrammatic representation of the logic for solving a task.
- Has boxes with lines connected.
- Boxes represent operations.
- Lines show the flow of control.
- Purpose:
  - To make the logic of the program clearer in a visual form.

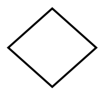
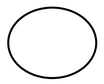
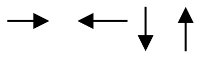
Need for Flowchart *(2 marks)*

- Process can be explained clearly through symbols and text in flow chart
- Provide effective program documentation and maintenance
- Easy to review and debug the program
- Logic of the program is communicated in much better way than algorithm
- Easy to understand

Flowchart symbols *(2 marks)*

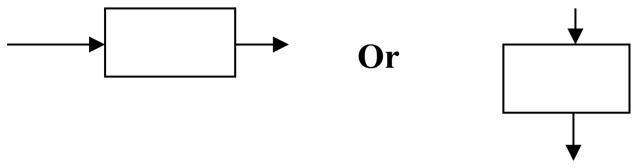
- Each symbol is for a specific purpose.
- Most commonly used flowchart symbols are,
  - Process
  - Decision
  - Data
  - Terminator
  - Connector
  - Flow lines

Symbol Name	Symbol	Description
Terminator		Start or end of the program
Process		Operation or action step
Data		Input or output operation

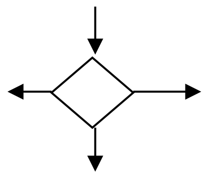
Decision		Decision making or branching
Connector		Connector or joining of two part of program
Flow lines		Indicate the direction of flow

<u>Rules (guide lines) for drawing flowchart</u> ***(2 marks)***

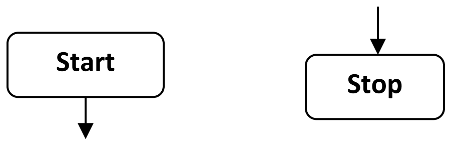
- Should have a start and end.
- Direction of flow must be from top to bottom and left to right.
- Relevant symbols must be used while drawing a flowchart.
- Should be clear, neat and easy to follow
- Only one flow line should come out from process symbol



- Only one flow line should enter a decision symbol



- Only one flow line is used in terminal symbol



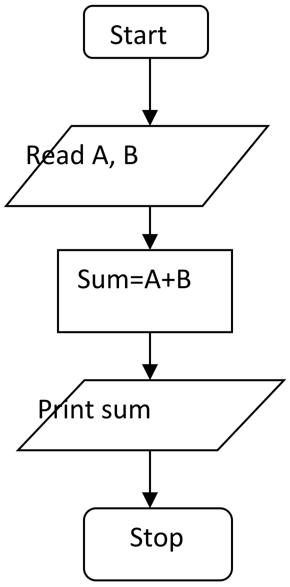
- For complex problem, connector symbol is used to reduce number of flow lines in flowchart.

<u>Control structure used in Flowchart</u> ***(16 marks & 8 marks)***

- There are three control structure used in flowchart, they are

1. **Sequence** ***(2 marks)***
    - In sequence control flow,
        - Instructions are executed in linear order
        - One after the other

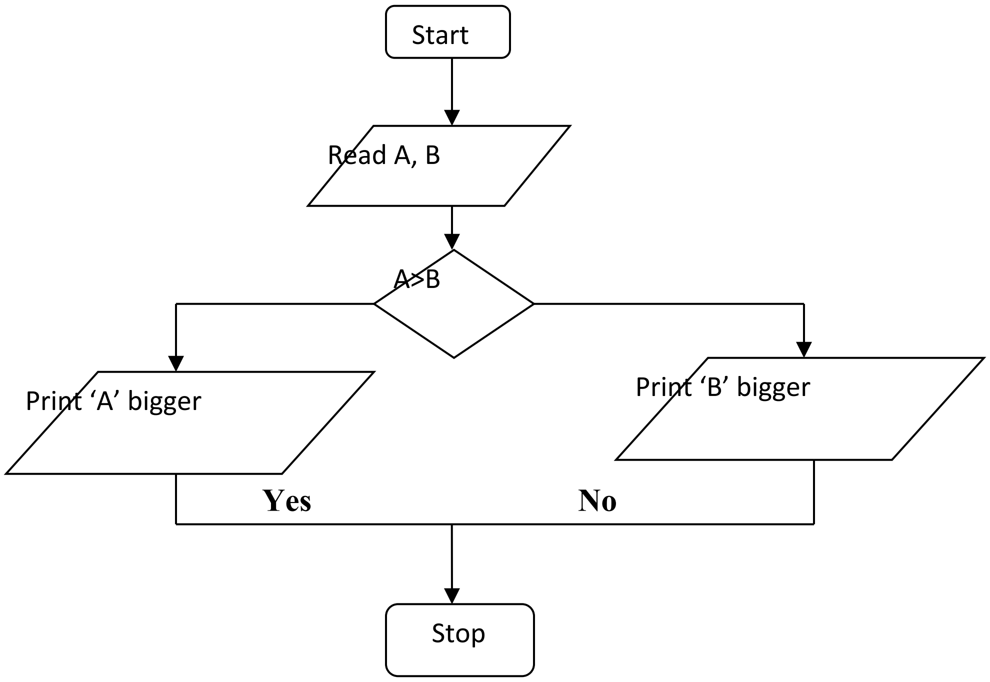
Example: flowchart to find sum of two numbers ***(2 marks)***



2. Selection (conditional) ***(2 marks)***

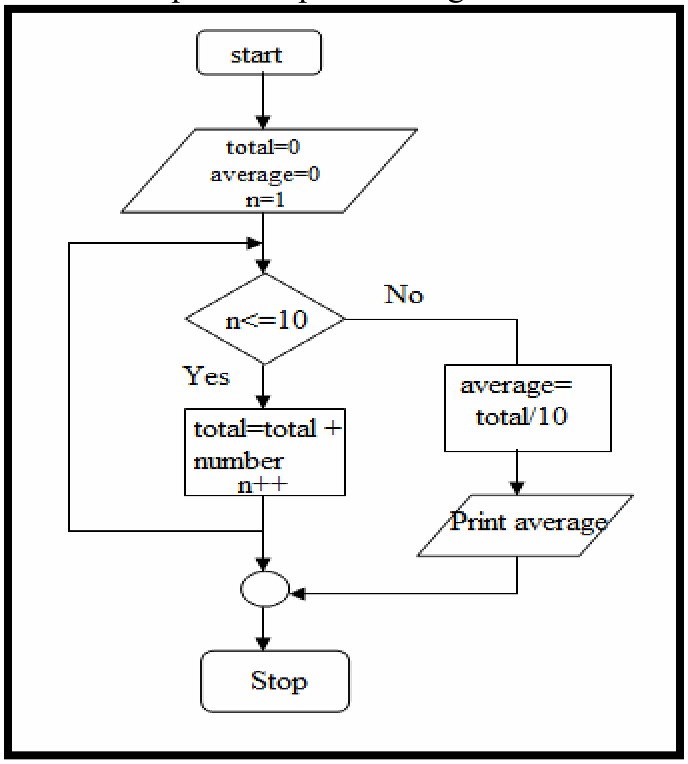
- In selection control flow,
  - Instructions are executed based on condition
  - If condition is true ,one path is followed
  - If condition is false, another path is followed

Example: flowchart to find biggest of two numbers ***(2 marks)***



3. **Iteration (looping)** ***(2 marks)***
   - In iterative control flow, a same set of instruction executed repeatedly until it satisfy particular condition

   Example: flowchart to compute and print average of 10 numbers ***(2 marks)***



<u>Advantage of Flowchart</u> *(2 marks)*

- Makes logic clear
- Visual representation
- Useful for coding
- Appropriate documentation.
- Act as a guide or blue print for program development
- Remove repeated and misplaced steps
- Increase in efficiency

<u>Disadvantage of Flowchart</u> *(2 marks)*

- Complex and long flowchart may run into multiple pages
- which is difficult to understand and follow
- difficult to modify
- Excessive use of the connector will confuse the programmers
- Cost of operation is high
- Updating is not regular

Difference between Algorithm, Flowchart and Pseudo code *(2 marks)*

Algorithm	Flowchart	Pseudo code
Algorithm is a sequence of instruction to solve a particular problem.	Flowchart is the graphical representation of algorithm.	Pseudo code is readable, English like representation of algorithm.
Can be represented using a flowchart or pseudo code.	It uses different kind of symbols for representation	It uses structured constructs of programming language for representation.
The user need knowledge to write algorithm for a task	User does not require any programming language knowledge to draw or understand a flowchart	User does not require any programming language knowledge to write or understand a pseudo code.

PROGRAMMING LANGUAGES *(8 marks & 2 marks)*

Programming language *(2 marks)*

- Computer programming languages are used to communicate instruction to a computer to perform a task.
- Programming language is like English language that can be understood by the computer through the translator of that language.
- It is based on certain syntactic and semantic rules
- Programming language forces the user to write very simple and exact instruction

Types of Programming language *(2 marks)*

- Programming language can be divided into three major categories
    1. Machine language
    2. Assembly language
    3. High level language(HLL)

1. **Machine language** ***(2 marks)***
    - It is native language of computer
    - It is in binary form
    - It uses only 0's and 1's to represent data and instruction
    - It is also called ***Low Level Language***(LLL)
    - It is very difficult to write or read instruction in binaries

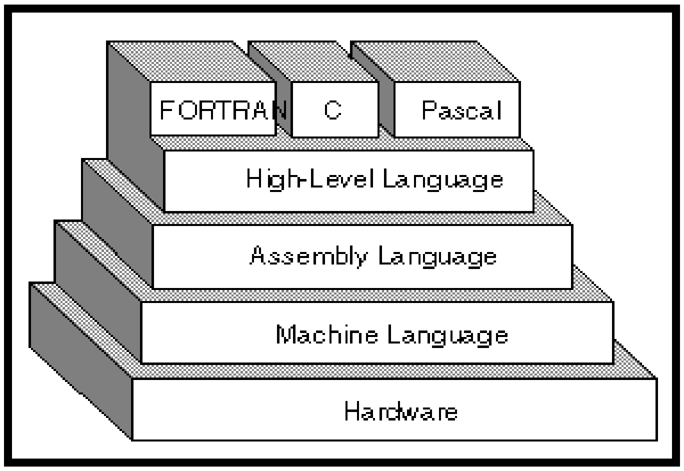
Example: 0100 00011001

Advantage

- Instruction can executed directly by the computer
- Fast in execution
- Bitwise operation is possible

Disadvantage

- It is machine dependent
- Not compactable
- Difficult to write
- Takes more time to write machine code



2. <u>Assembly language</u> ***(2 marks)***

- Instruction are written with mnemonics to simplify the program
- Use letters instead of 0's and 1's to run a machine
- The general format of an assembly instruction is

[Label] <opcode> <operands> [;Comments]

Example:

Label	**opcode**	**operands**	**Comments**
BEGIN	ADD	A, B	ADD B to A

- Mnemonics are converted to binaries with help of a translator known as ***assembler.***

Advantage

- Easy to understand and use
- Less error
- Faster
- More control on hardware

Disadvantage

- Machine dependent
- Harder to learn
- Less efficient
- No standardization
- No support for modern software technology

3. <u>High level language</u> ***(2 marks)***

- Instruction are written using English language with symbols and digits
- It is also called procedural language
- The commonly used high-level languages are FORTRAN, BASIC, COBOL, C, C++, Java etc.
- The high-level languages are converted to machine language using ***translators***
- **Types of translator**
    - Two types of translators are used, they are
        i) Compiler
        ii) Interpreter

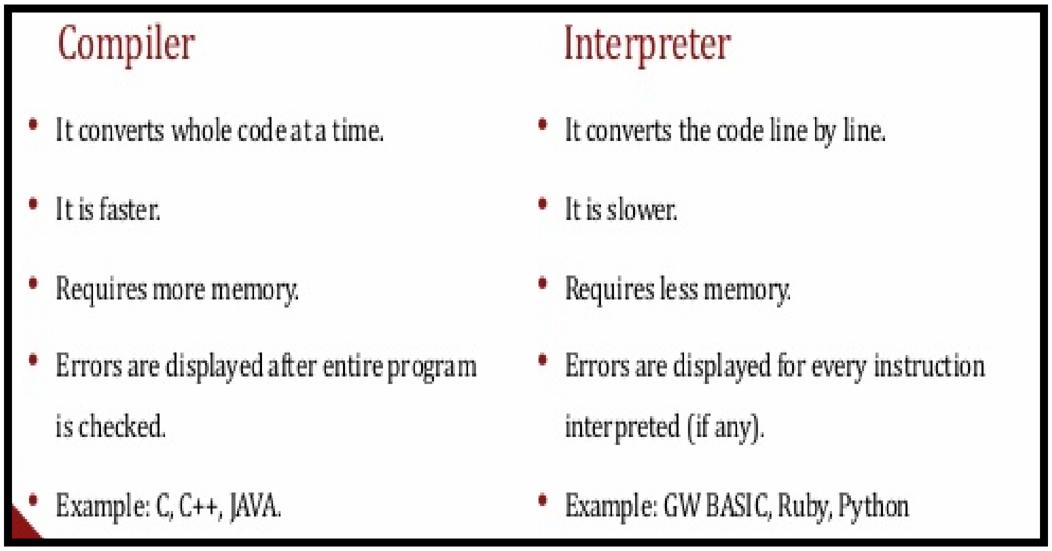
Compiler: (*2 marks)*

- Translate entire program of high-level languages into machine language
- Convert instruction from human understandable form to machine understandable form.
- Most of the languages use compiler
- It is faster

Interpreter: (*2 marks)*

- Translate each line of program instruction to machine language
- Slower
- Very few languages use interpreter

Difference between compiler and interpreter: (*2 marks)*



Compiler	Interpreter
It converts whole code at a time.	It converts the code line by line.
It is faster.	It is slower.
Requires more memory.	Requires less memory.
Errors are displayed after entire program is checked.	Errors are displayed for every instruction interpreted (if any).
Example: C, C++, JAVA.	Example: GW BASIC, Ruby, Python

Advantage

- Readability
- Machine independent
- Easy debugging
- Easier to maintain
- Low development cost

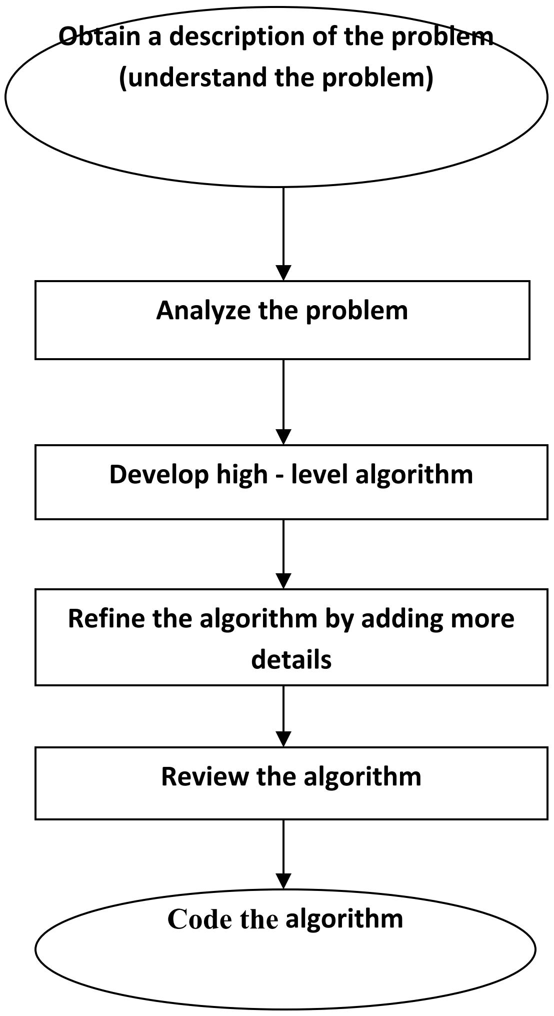
Disadvantage

- Poor control on hardware
- Less efficient

ALGORITHMIC PROBLEM SOLVING *(16 marks & 8 marks)*

Algorithmic Problem Solving

- An algorithm is a plan for solving a problem
- The development of an algorithm is the key step in algorithmic problem solving
- algorithmic problem solving is about the formation and solution of problems
- algorithmic problem solving process consist of six major steps, they are


Obtain a description of the problem (understand the problem)
Analyze the problem
Develop high - level algorithm
Refine the algorithm by adding more details
Review the algorithm
Code the algorithm

1. Obtain a description of the problem

- Algorithmic problem solving process starts by obtaining a description of the problem
- Understanding the problem
- Clarify the doubts after leading the problem description
- Identifying and collecting information
- Correct algorithm should work for all possible input

2. Analyze the problem

- Determine both the starting and the ending points for solving the problem
- Analyze both time and space efficiency
- Asking the following question often to determine the **starting point**
  - What data are available?

- Where the data from?
- What are the rules exist for working with the data?

- When determining the **ending point** algorithm needs to describe the characteristic of a solution.
- Asking the questions like
  - What new facts will we have?
  - What items would have changed?
  - What things will no longer exist?

3. Develop a high-level algorithm

- Algorithm is plan for solving a problem, but plans come in several levels of detail
- Developing high-level algorithm which includes major part of a solution
- Leaves the detail until later

4. Refine the algorithm

- Provide enough detail to solve the problem
- For complex problem go through the process several time, develop intermediate level algorithm
- Each time more details are added to the previous algorithm, for further refinement
- Stepwise refinement is the process of developing a detailed algorithm by gradually adding details to a high level algorithm

5. Review the algorithm

- Review the algorithm step by step to determine whether or not ,it will solve the original problem
- Asking these questions and seeking their answers is a good way to develop skills that can be applied to the next problem.

- ***Does this algorithm solve a very specific problem or general problem?***

  Example: for computing the area of a circle

  If radius is 5.2 meters (formula π*5.22) solves a very specific problem (formula π*R*R) solves a more general problem.

- ***Can this algorithm be simplified?***

  Example: One formula for computing the perimeter of a rectangle is:

  length + width + length + width

  A simpler formula would be:

2.0 * (length + width)

- ***Is this solution similar to the solution to another problem? How are they alike? How are they different?***

  Example: consider the following two formulae:

  Triangle area = 0.5 * base * height

  Rectangle area=length * width

  Similarities: Each computes an area. Each multiplies two measurements.

  Differences: Different measurements are used. The triangle formula contains 0.5.

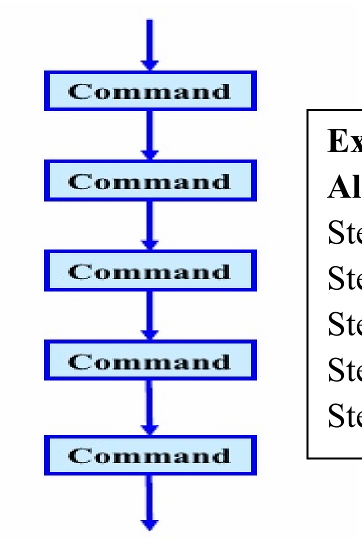
6. **Code the algorithm**
   - Once an algorithm developed ,it is coded using some programming languages
   - Each line of algorithm is translate to computer programs using programming languages
   - Executing the program
   - Checking errors and testing with different inputs

Simple strategies for Developing an Algorithm *(16 marks & 8 marks)*

Simple strategies for developing an Algorithm

- There are various kinds of algorithm developing technique formulated and used for different types of problems.
- They are,
  1. Sequence structure
  2. Selection structure
  3. Iteration structure
  4. Recursion structure

1. **Sequence (straight line) structure**
   - In sequence control flow,
     - Instructions are executed in linear order
     - One after the other
     - The logic flow is from top to bottom



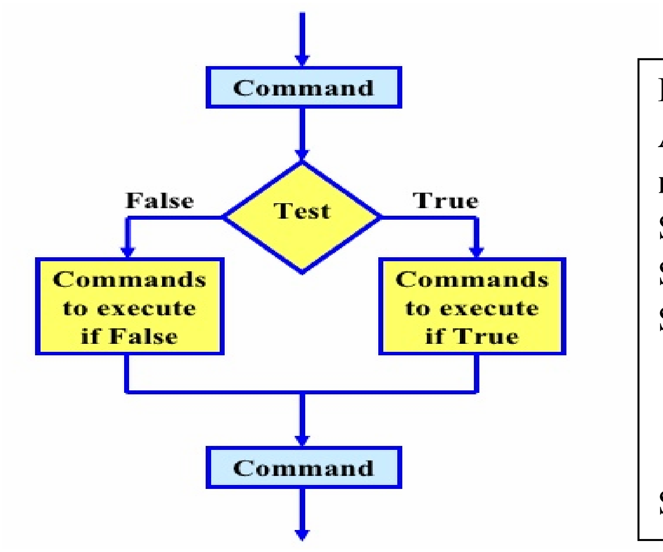
Example:

Algorithm to find sum of two numbers

```
Step 1: Start
Step 2: Read two numbers A and B
Step 3: Calculate sum= A + B
Step 4: Print the sum value
Step 5: Stop
```

-

2. **Selection (decision or branching)structure**
   - In selection control flow,
     - Instructions are executed based on condition
     - If condition is true ,one path is followed
     - If condition is false, another path is followed



Example:

Algorithm to find biggest of two numbers

```
Step 1: Start
Step 2: Read two numbers A and B
Step 3: IF (A>B) then
            Print 'A' is bigger
        ELSE
            Print 'B' is bigger
Step 4: Stop
```

- There are three selection structures in python, they are
  a) IF
  b) IF-ELSE
  c) ELIF

a) IF

- Check the condition
- If condition is true execute the statement
- If condition is false, the statement is not executed

b) IF-ELSE

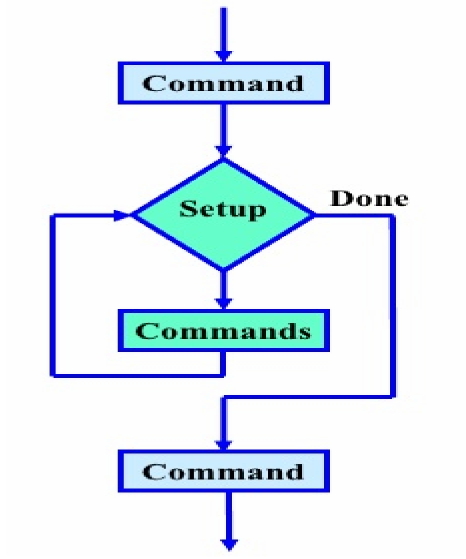
- Check the condition
- If condition is true execute the statement
- If condition is false, execute the else statement

c) ELIF

- Check multiple expressions
- Execute a block of code as soon as one of the conditions is true.

3. Iteration (looping) structure

- In iterative control flow, a same set of instruction executed repeatedly until it satisfy particular condition



Example:
Algorithm to compute and print average of 10 numbers

```
Step 1: Start
Step 2: total=0, average=0
Step 3: FOR 1 to 10
Step 4: Read number
Step 5: total=total + number
Step 6: END for
Step 7: average=total/10
Step 8: Print average
Step 9: Stop
```

- There are two repetition structures in python, they are
  a) WHILE
  b) FOR

a) WHILE

- It is also called entry controlled loop
- Check the condition
- If condition is true execute the body of the loop
- Process continues until the condition becomes false

b) FOR

- Consists of a header and a set of statements called the body
- Header contains information that controls the number of times that the body executes

4. **Recursion**

- In recursion technique a function or procedure is called by itself again and again until a given condition is satisfied.

Example: Recursive algorithm for finding the factorial of a number

```
Step 1: Start
Step 2: Read number n
Step 3: Call factorial (n)
Step 4: Print factorial f
Step 5: Stop
factorial (n)
Step 1: If n==1 then return 1
Step 2: Else
f=n*factorial (n-1)
Step 3: Return f
```

ILLUSTRATIVE PROBLEMS *(16 marks & 8 marks)*

1. <u>Find Minimum number in a list</u> *(8 marks)*

Problem statement

- To find minimum value in an list, take the first element and compare its value against the values of the other elements
- Repeat the process till the end of the list
- Finally the minimum value in the list is obtained

Algorithm

Step 1: Start
Step 2: Assign the first value of the list as minimum value
Step 3: Compare this value to the other values starting from second value
Step 4: When a value is smaller than the present minimum value is found, it becomes the new minimum
Step 5: Repeat the steps till the end of the list
Step 6: Print the minimum value
Step 7: Stop

Example

Consider a list

0	1	2	3	4
50	40	5	9	45

Step 1:

Take the first value in the list as Min value

0	1	2	3	4
50	40	5	9	45

min =50

Step 2:

Compare the Min value with next value in a list

40 < min value, so interchange min value

0	1	2	3	4
50	40	5	9	45

min =40

Step 3:

Compare the Min value with next value in a list

5 < min value, so interchange min value

0	1	2	3	4
50	40	5	9	45

min = 5

Step 4:

Compare the Min value with next value in a list

9 > min value, so need to interchange min value

0	1	2	3	4
50	40	5	9	45

min = 5

Step 5:

Compare the Min value with last value in a list

45 > min value, so need to interchange min value

0	1	2	3	4
50	40	5	9	45

min = 5

Finally the minimum value in this list is 5

2. <u>Insert a card in a list of sorted cards</u> *(8 marks)*

Problem statement

- To insert a card in the sorted card, increase the list size with 1.
- Insert new card in the appropriate position by comparing each element's value with the new one.
- When the position is found, move the remaining elements by one position up and the card can be inserted

Algorithm

Step 1: Start

Step 2: If it is the first element, it is already sorted

Step 3: Compare with all elements in the sorted sub-list

Step 4: Shift all elements in the sorted sub-list that is greater than the value

Step 5: Insert the value

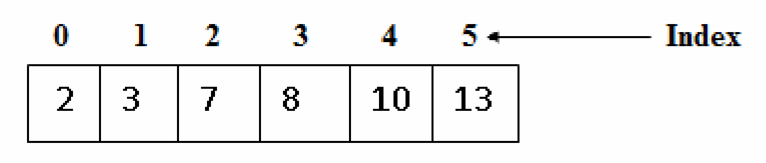
Step 6: Repeat until list is sorted

Step 7: Display all the card elements of a new list

Step 8: Stop

Example

Consider an array from index 0 through index 5 is already sorted

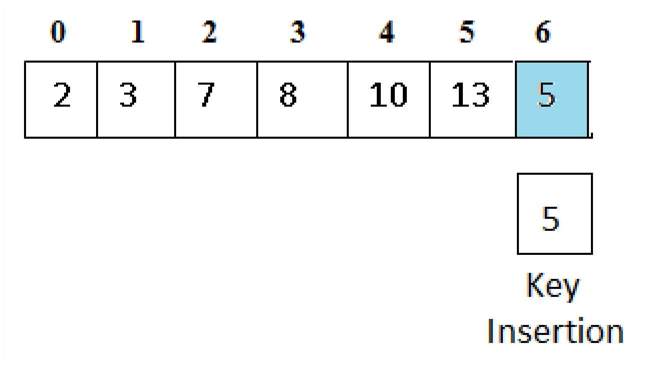


Insert new element-5 into the array

Need to insert the element currently in index 6 into this sorted array

Step 1:

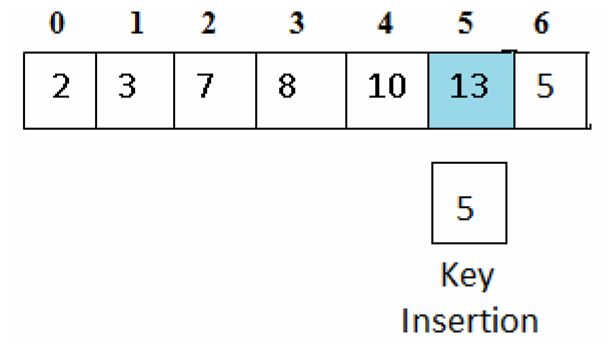
Store the element at index 6 in to variable called key



Step 2:

Compare key with the element at position 5.

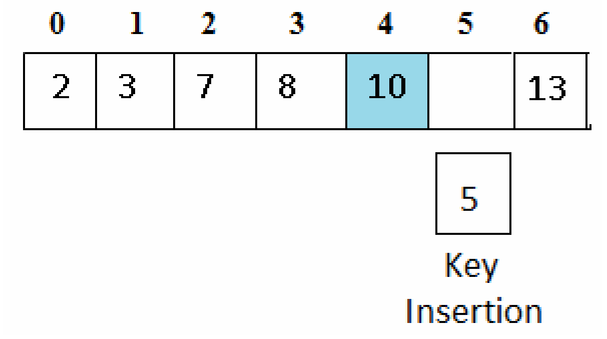
Key value (5) < Element at 5th position (13), so shift this element to position 6



Step 3:

Compare key with the element at position 4.

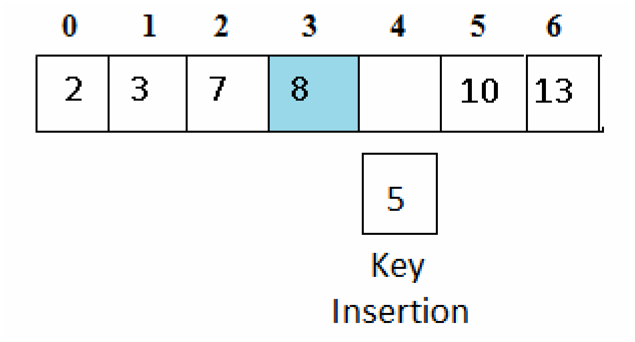
Key value (5) < Element at 4th position (10), so shift this element to position 5



Step 4:

Compare key with the element at position 3.

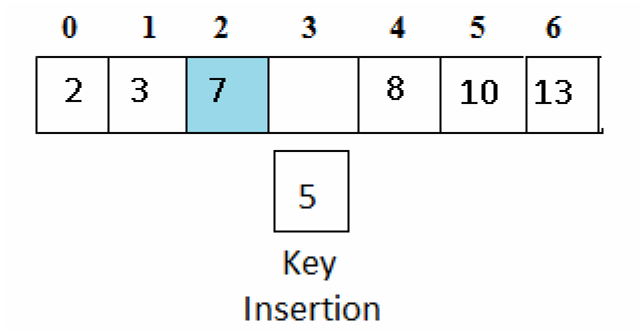
Key value (5) < Element at 3rd position (8), so shift this element to position 4



Step 5:

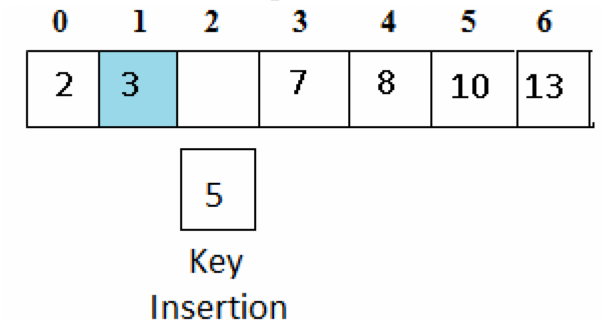
Compare key with the element at position 2.

Key value (5) < Element at 2nd position (7), so shift this element to position 3

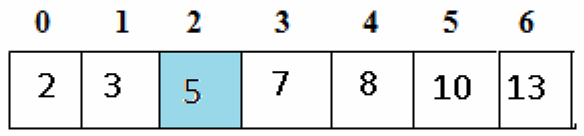


Step 6:

- Compare key with the element at position 1.
- Key value (5) > Element at 1st position (3), so no need to shift it over



- instead, drop key into the position immediately to the right of this element (that is, into position 2)



3. Guessing an integer number in a range *(8 marks)*

Problem statement

- Objective is to randomly generate integer number from 0 to n
- The player have to guess the number
- If player guess number correctly ,output an appropriate message
- If guess number is less than random number generated, output the message "Your guess is lower than the number. Guess again"
- Otherwise output the message "Your guess is higher than the number. Guess again"
- This process is repeated until the player enters the correct number

Algorithm

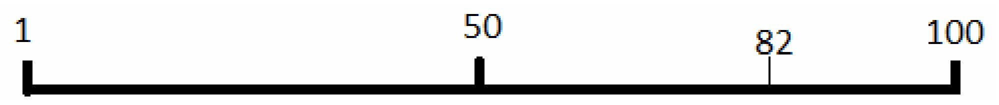
```
Step 1: Start
Step 2: Generate a random number and call it num
Step 3: Repeat the following steps until the player has guessed the correct number
        a. Enter the number to guess
        b. if(guess is equal to num)
           Print "You guessed the correct number"
           Otherwise
           if (guess is less than num)
           Print "You guess lower than the number.
                                                  Guess again!"
           Otherwise
           Print "You guess higher than the number.
                                                  Guess again!"
Step 4: Stop
```

Example

Consider a player want to guess a number between 1 and 100

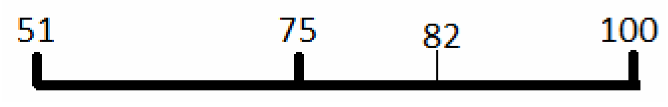
The number selected for guess is 82

Step 1: choose the middle number from the range 1 to 100, middle no is 50



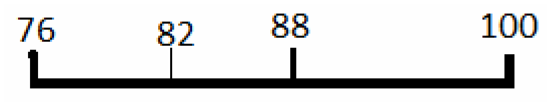
First guess: 50, here 50 is too low than guess number

Step 2: choose the middle number from the range 51 to 100, middle no is 75



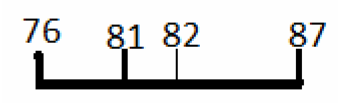
Second guess: 75, here 75 is too low than guess number

Step 3: choose the middle number from the range 76 to 100, middle no is 88



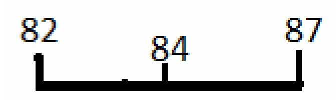
Third guess: 88, here 88 is too high than guess number

Step 4: choose the middle number from the range 76 to 87, middle no is 81



Fourth guess: 81, here 81 is too low than guess number

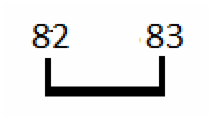
Step 5: choose the middle number from the range 82 to 87, middle no is 84



Fifth guess: 84, here 84 is too high than guess number

Step 6: choose the middle number from the range 76 to 87

Middle no is 82.5, which is rounded to 82



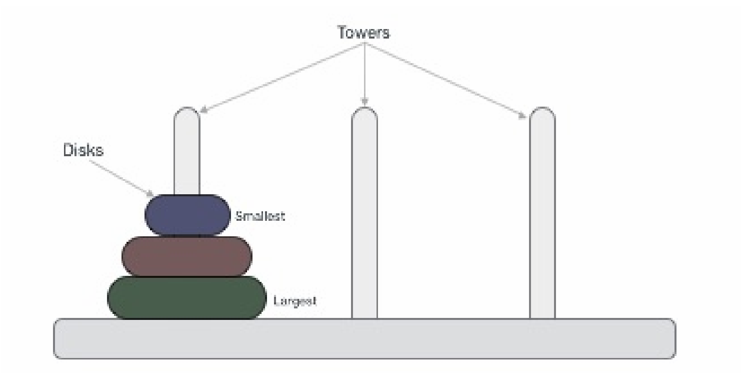
Sixth guess: 82, you guessed the correct number

4. Tower of Hanoi Problem (*8 marks)*

Problem statement

- Tower of Hanoi is one of the classical problems of computer science
- The problem states that,
    1. There are three stands(Stand 1,2 and 3) on which a set of disks, each with a different diameter are placed
    2. Initially, the disks are stacked on Stand 1,in order of size with the larger disk at the bottom

The initial structure of Tower of Hanoi with three disks is shown below



Tower of Hanoi with three disks

- **Rules to move the disk**
    - The 'Tower of Hanoi problem' is to find the sequence of disk moves so that all the disks moved from stand-1 to stand-3 using the following rules
        1. Move only one disk at a time
        2. A larger disk cannot be placed on top of a smaller disk.
        3. Only the "top" disk can be removed

- The recurrence relation for solving the Tower of Hanoi problem can be written as
  Tower of Hanoi(disks)=

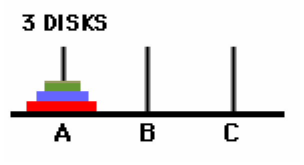
Algorithm

```
Step 1:  START
Step 2:  Procedure Hanoi (disk, source, dest, aux)
Step 3:  If disk == 0 THEN
                  Step 3.1:  move disk from source to dest
Step 4:  ELSE
                  Step 4.1:  Hanoi (disk-1, source, aux, dest)
                  Step 4.2:  move disk from source to dest
                  Step 4.3:  Hanoi (disk-1, aux, dest, source)
Step 5:  END If
Step 6:  END Procedure
```

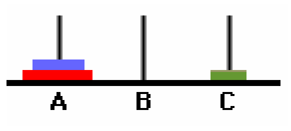
Step 7: STOP

Example

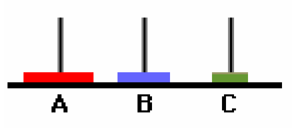
- The general strategy for solving the Tower of Hanoi problem with three disks as follows



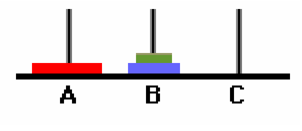
Move 1: move disk 3 to post C



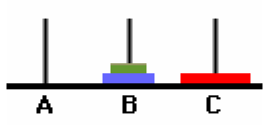
Move 2: move disk 2 to post B



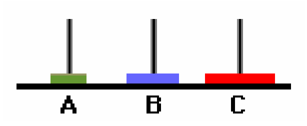
Move 3: move disk 3 to post B



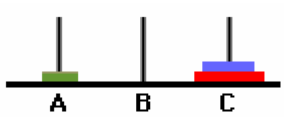
Move 4: move disk 1 to post C



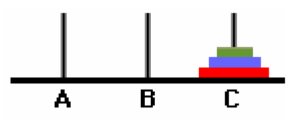
Move 5: move disk 3 to post A



Move 6: move disk 2 to post C



Move 7: move disk 3 to post C



IMPORTANT QUESTIONS

PART-B

1. Define algorithm? Explain the characteristics, benefits, limitation of an algorithm in detail. ******* ***(16 marks)***
2. Explain the building blocks of algorithm in detail ******* ***(16 marks)***
3. Define flowchart? Explain the guidelines for preparing flowcharts, benefits, limitation of flowchart in detail. ******* ***(16 marks)***
4. Define pseudocode? Explain the guidelines for preparing pseudocode, benefits, limitation of pseudocode in detail. ******* ***(16 marks)***
5. Explain algorithm, flowchart, Pseudocode in detail? ******* ***(16 marks)***
6. Define programming languages? Explain the various types of programming languages in detail******* ***(16 marks)***
7. Explain the steps involved in algorithmic problem solving techniques in detail *****(***8 marks)***
8. Explain different strategies used for developing algorithm in detail *****(***16 marks)***

   (Or)

   What is the basic logic structures used in writing structured program?
9. Explain the algorithm for Tower of Hanoi problem with example in detail ***(16 marks)***
10. Explain the algorithm for guessing an number in a range with example in detail ***(16 marks)***
11. Explain the algorithm to find minimum number in a list with example in detail ***(16 marks)***
12. Explain the algorithm to insert a card in list of sorted list with example in detail ***(16 marks)***

PART-A

1. Define computer
2. Define program *
3. What are the steps involved in problem solving methodology (software life cycle)?
4. Define coding
5. Define debugging
6. Define algorithm ***
7. List out the characteristics of an algorithm ***
8. What are the qualities of good algorithm? ****
9. List out the building blocks of algorithm
10. List out the advantage and disadvantages of an algorithm ***
11. Difference between algorithm and flow chart
12. Define pseudo code ***
13. What are keywords used in pseudo code?

14. What are the rules for writing pseudo code? ***
15. List out the advantage and disadvantages of a pseudo code? ***
16. Difference between algorithm and pseudo code
17. Define Flow chart ***
18. List out the needs for flow chart. ***
19. What are the guidelines for drawing flow chart? ***
20. List out the advantage and disadvantages of a flow chart ***
21. Difference between algorithm and pseudo code
22. Define programming languages? List its types **
23. Define Machine language **
24. Define assembly language
25. Define high level language
26. What is meant by translator? List its types
27. Define compiler **
28. Define interpreter **
29. Difference between compiler and interpreter **
30. What are the steps involved in algorithmic problem solving?
31. List out the techniques used for developing algorithms
32. Define iteration(looping) **
33. Define selection(branching)
34. Define recursion with example. ***

FIRST PEFERENCE

1. **ALGORITHM-DEFINITION,CHARACTERISTICS,BUILDING BLOCKS,ADVANTAGE & DISADVANTAGE ****
2. **FLOW CHART- DEFINITION,NEED,SYMBOLS,GUIDELINES, ADVANTAGE & DISADVANTAGE *****
3. **PSEUDOCODE- DEFINITION, GUIDELINES ,ADVANTAGE & DISADVANTAGE *****
4. **TOWER OF HANOI****
5. **GUESSING NUMBER IN A RANGE****
6. **FINDING MINIMUM IN A LIST****
7. **INSERT A CARD IN A LIST OF SORTED CARD****
8. **STEPS IN ALGORITHMIC PROBLEM SOLVING TECHNIQUES*****
9. **DIFFERENT STRATEGIES FOR DEVELOPING ALGORITHM*****
10. **PROGRAMMING LANGUAGES,TYPES******

Important Examples -Algorithm, Flow chart & Pseudo code *(16 & 8 marks)*

Algorithm, Flow chart, Pseudo code for the following

- Greatest of three numbers
- Roots of quadratic equation
- Factorial of a number
- Fibonacci series
- Prime number or not
- Swapping two variables without third variables
- GCD of two numbers

*********ALL THE BEST***********

Roots of quadratic equation

Algorithm

```
Step 1: Start
Step 2: Read the value of a, b, c
Step 3: d = b*b - 4*a*c
Step 4: if d < 0 then
            Display the Roots are Imaginary.
Step 5: else if d = 0 then
            Display Roots are Equal.
            r = -b / 2*a
            display r
Step 6: else
            r1 = -b + √d / 2*a
            r2 = -b - √d / 2*a
            display Roots are real and r1, r2
Step 7: Stop
```

Pseudo code

```
START
READ the value of a, b, c
CALCULATE d = b*b - 4*a*c
IF d < 0 THEN
    DISPLAY the Roots are Imaginary.
ELSE if d = 0 then
    DISPLAY Roots are Equal.
    DISPLAY, r = -b / 2*a
ELSE
    r1 = -b + √d / 2*a
    r2 = -b - √d / 2*a
    DISPLAY Roots are real and r1, r2
STOP
```

Flow chart – Refer text book pg.no 41