Unit 1
Algorithmic Problem Solving

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

1.1 INTRODUCTION TO COMPUTER PROGRAMMING

A computer is a useful tool to solve a variety of problems. A computer program is written to make the computer to do a task. A computer program is a list of instructions that performs a specific task when executed by a computer. A computer program instructs a computer to do a specific task. As computer is an electronic machine, it can feel only electricity. It cannot understand human languages. Computers can understand a language that is related to electricity i.e.,. Binary language or machine language. The binary language has only two numbers 0's and 1's where 0 represents low voltage and 1 represents high voltage. Words are formed by combining 1's and 0's. It is too difficult for humans to write instructions in a pure binary form. There comes the development of Assembly language and Programming languages. A computer program is written in a programming language. A programming language is a notation designed to connect instructions to a machine or a computer. A programming language is a computer language consists of instructions for a computer. Most popular programming languages are C, C++ and Java. Python,

1.2 ALGORITHM

The sequence of steps to be performed in order to solve a problem by the computer is known as an algorithm.

Programs = Algorithms + Data

Another way to describe algorithm is the sequence of unambiguous instructions. It starts from an initial input of instructions that describe a computation that proceeds through a finite number of well-defined successive steps, producing an output and a final ending state. Algorithms was first developed by Persian scientist, astronomer and mathematician Abdullah Muhammad bin Musa al-Khwarizmi in 9th century. He was often cited as "The father of Algebra", and was responsible for the creation of the term "Algorithm".

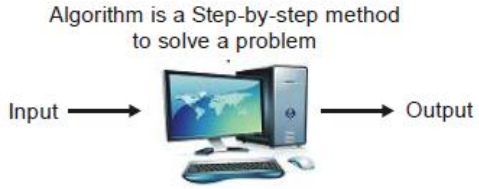


Figure 1.1 Definition of the Algorithm

An algorithm is a form that embeds the complete logic of the solution. Its proper written form is called a program, or code. Thus, algorithmic problem solving actually comes in two phases:

- Derivation of an algorithm that solves the problem.
- Conversion of the algorithm into program code.

The first phase is the algorithmic phase and the second phase is the coding phase. The coding phase is comparatively easier, as the logic is already explained in the algorithmic phase. Coding phase ensures that the syntax rules of the programming language are adhered. The algorithmic phase is the difficult phase, for two main reasons.

- Firstly, it challenges the mental facilities to search for the right solution.
- Secondly, it requires the ability to articulate the solution concisely into step by-step instructions, a skill that is acquired only through lots of practice.

Algorithms are converted into programs, are the software. The machine that runs the programs is the hardware.

Table 1.1 Comparison between Computer Hardware and Software

Description	Hardware	Software
Definition	Devices that are required to store and execute the software.	Computer Software is a set of instructions for a computer to perform specific operations.
Types	Input Devices, Output Devices, Storage Devices, Processing Devices, Control Devices.	System Software, Programming Software, Application Software.
Example	CD-ROM, Keyboard, Monitor, Printer, Scanner.	Adobe Acrobat, Microsoft Word
Nature	Hardware is physical in nature.	Hardware is logical in nature.

For each problem, there may be many different algorithms. For each algorithm, there may be many different implementations (programs).

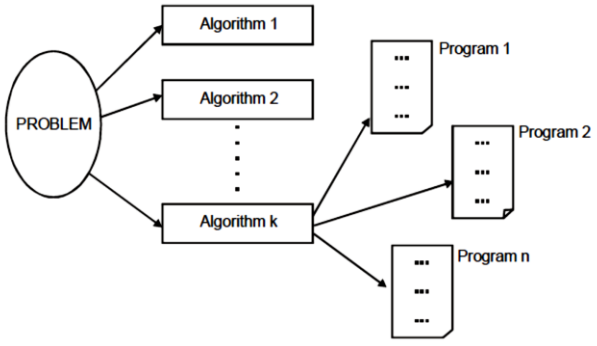


Figure 1.2 Problems vs Algorithms vs Programs

During the process of solving any problem, one tries to find the necessary steps to be taken in a sequence. For Example, if you want to talk to your friend, you need to follow the following steps.

Algorithm: Calling a friend on the telephone

Input: The telephone number of your friend.

Output: Talk to your friend

Steps:

1. Pick up the phone and listen for a dial tone.
2. Press each digit of the phone number on the phone.
3. If busy, hang up phone, wait 2 minutes, jump to step 2.
4. If no one answers, leave a message then hang up.
5. If no answering machine, hang up and wait 2 hours, then jump to step 2.
6. Talk to friend.
7. Hang up phone.

1.2.1 Algorithm Development Process

An algorithm is a plan for solving a problem. There are many ways to write an algorithm. Some are very informal, some are quite formal and mathematical in nature, and some are quite graphical. The form is not particularly important as long as it provides a good way to describe and check the logic of the plan. The development of an algorithm (a plan) is a key step in solving a problem. All programming languages share basic code constructs; statements, loops (do, for, while), flow-control (if-else) etc. These common constructs can be used to write an algorithm. There are many ways to write algorithms. You can have your own way of writing algorithms but you must be consistent. The algorithm must take up a well definite finite set of input and produce a well definite finite set of output. The development of an algorithm (a plan) is a key step in solving a problem. Once the algorithm is ready, it can be translated it into a computer program in some programming language. The algorithm development process consists of five major steps.

PROGRAM 1

Step 1: Obtain a description of the problem.

Step 2: Analyse the problem.

Step 3: Develop a high-level algorithm.

Step 4: Refine the algorithm by adding more detail.

Step 5: Review the algorithm.

Step 1 Obtain a description of the problem

The problem should be clearly explained, so that it's easy for the developer to find the solution for the problem. The problem description suffers from one or more of the following types of flaws:

- The description relies on unstated assumptions.
- The description is ambiguous.
- The description is incomplete.
- The description has internal contradictions.

These flaws are seldom due to carelessness by the client. or sometimes by natural languages (English, French, Korean, etc.)

Step 2 Analyse the problem

The purpose of this step is to determine both the starting and ending points for solving the problem. When determining the starting point, start with following questions:

- What data are available?
- Where is that data?
- What formulas are related to the problem?
- What rules are needed for the data?
- What relationships exist among the data values?

When determining the ending point, the characteristics of a solution are described. The following questions help in determining the ending point.

- What new facts will arrive?
- What items will change?
- What things will no longer exist?

Step 3 Develop a high-level Algorithm

An algorithm is a plan for solving a problem. It's usually better to start with a high-level algorithm that includes the major part of a solution, but sometimes more details can be added later. An example is given to demonstrate a high-level algorithm.

Problem Statement: I need to make a tea.

Analysis: I don't have milk.

High-level algorithm:

- Go to a stores that sells milk.
- Purchase milk and come home.
- Prepare Tea.

Though this algorithm seems to be satisfactory, it lacks many details such as the following.

- Which store I need to visit?
- Which milk product I need to buy?
- How I go to the stores: walk, drive, ride my two-wheeler, take the bus.

These kinds of details are considered in the next step of our process.

Step 4 Refine the algorithm by adding more detail

A high-level algorithm shows the major steps that need to be followed to solve a problem. Our goal is to develop algorithms that lead to computer programs. In this case, consider all capabilities of the computer and provide enough details so that it's easy to write algorithms. In simple examples moving from high-level to a detailed algorithm is done in a single step, but this is not always reasonable. For larger, more complex problems, it is common to go through several times. Each time, more details are added to the previous algorithm. This technique of gradually working from a high level to a detailed algorithm is often called stepwise refinement. Stepwise refinement is a process for developing a detailed algorithm by gradually adding detail to a high-level algorithm.

Step 5 Review the Algorithm

The final step is to review the algorithm. Check the algorithm step by step to determine whether it will solve the original problem or not.

- Does this algorithm solve a very specific problem or does it solve a more general problem?

- If it solves a very specific problem, should it be generalized?

Example:

Compute the area of a circle having radius 5.2 meters.

If the algorithm is having the formula, □ *5.22 then it solves a very specific problem.

But the algorithm computes the area of any circle, □ *r2 then it solves a more general problem.

1.2.2 Properties of an Algorithm

Finiteness:

- The algorithm must always terminate after a finite number of steps.

Definiteness:

- Each instruction must be clear, well-defined and precise.
- There should not be any ambiguity.

Effectiveness:

- Each Instruction must be simple and be carried out in a finite amount of time.

Input:

- An algorithm has zero or more inputs, taken from a specified set of objects.

Output:

- An algorithm has one or more outputs, which have a specified relation to the inputs.

Feasibility:

- It must be possible to perform each instruction.

Generality:

- The algorithm must be able to work for a set of inputs rather than a single input.

1.3 BUILDING BLOCKS OF ALGORITHMS

It has been proven that any algorithm can be constructed from just three basic building blocks. These three building blocks are instructions: state, control flow, and functions.

Building Block	***Common name***
Instruction	Action/Sequence
State/Selection	Decision
Control Flow/Iteration	Repetition or Loop

1.3.1 Instruction/ Sequence

Sequence is the specific order in which instructions are performed in an algorithm. This means that the computer will run the codes in order, one line at a time from the top of the program to the bottom of the program. When an instruction is executed the execution control moves to the next immediate step. The sequence is exemplified by sequence of statements placed one after the other the one above or before another gets executed first. It will start at line 1, and then execute line 2 then line 3 and so on till it reaches the last line of the program.

Representation

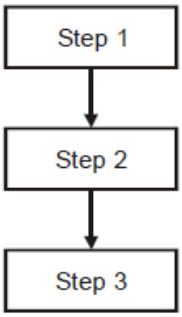


Figure 1.3 Sequential Statement

Example

Making Pizza: base -> tomato sauce -> cheese -> bake

The algorithm given in below example, the steps are executed one after the other in sequential order. Hence all the steps are instructions. An instruction can be anyone of the following types;

Computational: Any instruction that does computations on the inputs or other data to derive the output or another data.

Assignment: Any instruction that assigns value to a variable.

Input: Any instruction that gets the input values.

Output: Any instruction that displays or prints the input values.

Problem - Find the sum, average and product of three numbers.

Read X, Y	**Input**
Set Z as 100	**Assignment**
Compute Sum (S) as X + Y + Z	
Compute Average (A) as S / 3	**Computational & Assignment**
Compute Product (P) as X * Y * Z	
Display the Sum, Average and Product	**Output**

1.3.2 State/Selection

Selection is a decision or question. Sometimes we want only some lines of code to be run only if a condition is met, otherwise you want the computer to ignore these lines and jump over them. A state is the result of a test or condition - either True or False. If the result is True you take a certain course of action and if the result is False you take another course of action.

This is referred as if-then-else statement:

If Condition A is True then
    perform Action X
else
    perform Action Y.

Representation

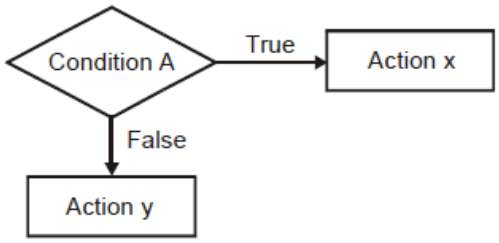


Figure 1.4 State/Selection Statement

Example:

Making Pizza: base -> tomato sauce -> topping? (if yes, add parsley. if not, skip) -> cheese -> bake.

PROGRAM 2

Problem - find out number is odd or even.

Step 1 : Start
Step 2 : Input number
Step 3 : rem=number mod 2
Step 4 : IF rem=0 THEN
Print "number even"
ELSE
Print "number odd"
END IF
step 5 : stop

There can be any number of conditions in a single State. Moreover a state can be placed subsequent of another state. The step 3 in the following method is a state with many conditions and also the Else is followed by another If.

Problem - find the smallest of three given numbers

Step 1: Start
Step 2: Get three numbers A, B & Y
Step 3: If (A < B and A < C) Then
Display A as the smallest number
Else If (B < A and B < C) Then
Display B as the smallest number
Else
Display C as the smallest number
End If
Step 4: Stop

1.3.3 Control Flow

A control flow statement is a statement whose execution results in a choice being made as to which of two or more paths to follow.

Iteration is a type of Control Flow Statement. Iteration is used in computer programs to repeat a set of instructions. There are two types of iteration.

Count controlled iteration will repeat a set of instructions a specific number of times.

Condition controlled iteration will repeat the instructions until a specific condition is met.

Representation

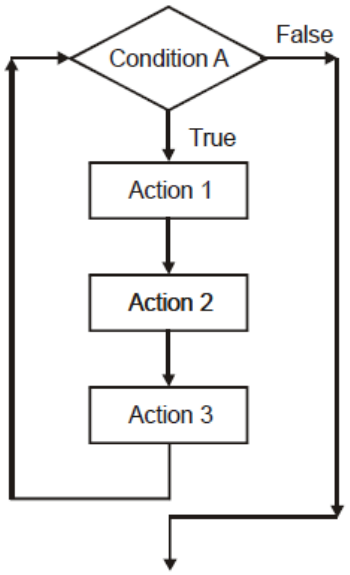


Figure 1.5 Structure of Control flow

Without Iteration

A simple algorithm can be created for cleaning teeth. Suppose a person has ten top teeth. To make sure that every one of the top teeth is cleaned, the algorithm is something like this:

Step 1: Put toothpaste on toothbrush
Step 2: Clean tooth 1 using toothbrush
Step 3: Clean tooth 2 using toothbrush
Step 4: Clean tooth 3 using toothbrush
Step 5: Clean tooth 4 using toothbrush
Step 6: Clean tooth 5 using toothbrush
Step 7: Clean tooth 6 using toothbrush
Step 8: Clean tooth 7 using toothbrush
Step 9: Clean tooth 8 using toothbrush
Step 10: Clean tooth 9 using toothbrush
Step 11: Clean tooth 10 using toothbrush
Step 12: Rinse toothbrush

PROGRAM 5

Steps 2 through to 11 are repeated, just cleaning a different tooth every time. Iteration can be used to simplify the algorithm.

With Iteration

But, how do we know that all teeth are clean. A condition is needed to solve this problem. The condition, in this case, is to check whether the number of teeth cleaned equals ten. If the condition is True ie.the number of teeth cleaned equals ten, then no more iterations is needed. If the condition is False ie.the number of teeth cleaned is less than ten, then another iteration occurs.

Step 1: Put toothpaste on toothbrush

Step 2: Clean tooth 1 using toothbrush
Step 3: Move to next tooth
Step 4: Repeat steps 2 and 3 until all teeth are clean
Step 5: Rinse toothbrush
The counter is called number_of_teeth_cleaned, and the condition is 'if number_of_teeth_cleaned< 10'.

With conditional Iteration

Step 1: set number_of_teeth_cleaned to 0
Step 2: Put toothpaste on toothbrush
Step 3: Clean a tooth using toothbrush
Step 4: Increase number_of_teeth_cleaned by 1
Step 5: If number_of_teeth_cleaned< 10 then go back to step 3
Step 6: Rinse toothbrush

1.3.4 Functions

Any complex problem will become simpler if the problem is broken smaller and the smaller problems are solved. The functions are solutions to smaller problems. These can be used to solve bigger, complex problems. Algorithm for an online shopping system will have several lines. The entire problem of solving the online shopping system can be broken down into smaller functional aspects called modules. Few modules can be used in several other modules. Hence writing functions would increase the readability and efficiency of the algorithm. Considering the online shopping system, the products and its details can be segregated as the following;

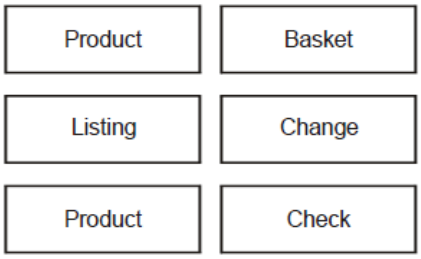


Figure 1.6 Functions of online shopping system

Algorithm to find the prime numbers in a range of numbers.

IsPrime(No)
Begin
    Check if No is divisible by any other number between 2 and No-1
    If divisible return False
    Else return True
End
Step 1: Get the Start and End of range
Step 2: Let N = Start
Step 3: Check if number N is prime using IsPrime(N)
Step 4: Print number if True is returned
Step 5: Increment N by 1
Step 6: Repeat steps 3 and 5 Until End is reached

1.4 NOTATIONS

Algorithms can be expressed in many different notations, including Natural Language, pseudo code, flowcharts and programming languages. Natural language tends to be verbose and ambiguous, and is rarely used. Pseudo code and flowcharts are structured ways to express algorithms. Pseudo-code is something that represents the algorithm through structured human language. Flowchart is something that represents the algorithm graphically. Programming languages are intended for expressing algorithms in a form that can be executed by a computer.

1.4.1 Pseudo Code

Pseudocode is an informal language used by programmers to develop algorithms. Pseudocode is a "text-based" detail design tool. The name comes from the word "Pseudo" that means imitation or false and "code" refers to the instructions written in a programming language. The Pseudocode is readable only when the lines/statements that are dependent are indented properly. An algorithm gives the logic to solve a given problem. However the Pseudocode is a set of statements in plain English which may be translated later into a programming language. Algorithm forms the stage one and Pseudocode forms the second stage, just before implementation of a program.

Rules to follow when writing pseudocode:

- Only one statement per line
  - Readability improves if just one action for the computer is written in one statement.
- Capitalized initial keyword
  - Keywords like READ, WRITE, etc are in caps.
- Indent to show hierarchy.
  - In loops, states and iterations the logically dependent statements must be indented
- End multi-line structures.
  - To improve readability the initial start and end of the several lines must be specified properly
- Keep statement language independent.
  - The programmer must never use the syntax of any programming language

Express an algorithm to get two numbers from the user (dividend and divisor), testing to make sure that the divisor number is not zero, and displaying their quotient using pseudocode

1. Declare variables: dividend, divisor, quotient
2. Prompt user to enter dividend and divisor
3. Get dividend and divisor
4. IF divisor is equal to zero, THEN
    4.1. DO
        4.1.1. Display error message, "divisor must be non-zero"
        4.1.2. Prompt user to enter divisor
        4.1.3. Get divisor
    4.2. WHILE divisor is equal to zero
5. ENDIF
6. Display dividend and divisor

7. Calculate quotient as dividend/divisor
8. Display quotient

Write an algorithm to determine a student's final grade and indicate whether he is getting pass mark or fail mark. The final grade is calculated as the average of three marks.

Pseudo code:

- Input a set of 3 marks
- Calculate their average by summing and dividing by 3
- if average is below 50
    - Print "FAIL"
- else
    - Print "PASS"

Algorithm

Step 1: Input Mark 1, Mark 2, Mark 3
Step 2:
Average = (Mark1 + Mark2 + Mark3)/3
Step 3: if (Average < 50) then
Print "FAIL"
else
Print "PASS"
endif

1.4.2 Flowcharts

A flowchart is a graphical representation of an algorithm. These flowcharts play a vital role in the programming and are quite helpful in understanding the logic of complicated problems. Flowcharts were introduced by Frank Gilberth in 1921, and they were called "Process Flow Charts" at the beginning. A flowchart is a diagram made up of boxes, diamonds and other shapes, connected by arrows - each shape represents a step in the process, and the arrows show the order in which they occur. Once the flowchart is drawn, it becomes easy to write the program in any high level language.

1. Types of Flowcharts

There are many different types of flowcharts. There are flowcharts for analysts, designers, managers or programmers for their understanding.

- *Document flowcharts* have the purpose of showing existing controls over a document-flow through the components of the system. This chart is read from left to right and explain the flow of documents through various business units.
- *Data flowcharts* have the purpose of showing the controls governing data flows in the system. Data flowcharts are used primarily to show how data is transmitted through the system rather than the control flow.
- *System flowcharts* shows the controls located at the physical or resource level. System flowcharts show the flow of data through the major components of the system such as data entry, programs, storage media, processors and communication networks.

- *Program flowcharts* shows the controls placed internally to the program within a system. Program flowchart is a gift to programmers as it makes programming task very easy and systematic

2. Flowcharting Symbols

- Process / Operation Symbols
- Branching and Control of Flow Symbols
- Input and Output Symbols
- File and Information Storage Symbols
- Data Processing Symbols

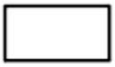
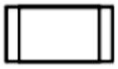
Symbol	*Name*	*Description*
	**Process**	**Show a Process or action step. This is the most common symbol used in flowchart.**
	**Predefined Process (Subroutine)**	**A Predefined Process symbol is a marker for another process or series of process. This shape commonly depicts sub-processes or subroutines.**
	**Alternate Process**	**It is used when the process flow step is an alternate to the normal process step.**
	**Delay**	**The Delay flowchart symbol depicts any waiting period that is part of a process.**
	**Preparation**	**Any process step that is a Preparation process flow step, such as a set-up operation.**
	**Manual Operation**	**Manual Operations flowchart shapes show which process steps are not automated.**

Table 1.2 Process / Operation Symbols in Flowchart

Symbol	*Name*	*Description*
	Flow Line (Arrow, Connector)	Flow line connectors show the direction that the process flows.
	Terminator (Terminal Point, Oval)	Terminators show the start and stop points in a process.
	Decision	Indicates a question or branch in the process flow.
	Connector (Inspection)	It is used to show a jump from one point in the process flow to another.
	Off-Page Connector	Off-Page Connector shows continuation of a process flowchart onto another page.
	Merge (Storage)	This symbol shows the merging of multiple processes or information into one.
	Extract (Measurement)	This symbol shows when a process splits into parallel paths.
	Or	The logical Or symbol shows when a process diverges - usually for more than two branches.
	Summing Junction	The logical Summing Junction flowchart shape is shows when multiple branches converge into a single process.

Table 1.3 Branching and Control of Flow Symbols in Flowchart

Symbol	Name	Description
	Data (I/O)	The Data flowchart shape indicates inputs to and outputs from a process.
	Document	Document flowchart symbol is for a process step that produces a document.
	Multi-Document	Multi-Document flowchart symbol is for a process step that produces a multiple documents.
	Display	Indicates a process step where information is displayed.
	Manual Input	Manual Input flowchart gets the input manually into a system.
	Card	This symbol is companion to the punched tape flowchart shapes. This shape is seldom used.
	Punched Tape	Punched Tape symbol is used for input into old computers and CNC machines.

Table 1.4 Input and Output Symbols in Flowchart

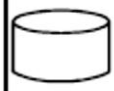
Symbol	Name	Description
	Stored Data	A general Data Storage flowchart shape used for any process step that stores data.
	Magnetic Disk Database	The most universally recognizable symbol for a data storage location, this flowchart shape depicts a database.
	Direct Access Storage	Direct Access Storage is a like Hard Drive.
	Internal Storage	It is used in programming flowcharts to store information in memory.
	Sequential Access Storage (Magnetic Tape)	Although it looks like a 'Q', the symbol is supposed to look like a reel of tape.

Table 1.5 File and Information Storage Symbols in Flowchart

Symbol	Name	Description
	Collate	The Collate flowchart shape indicates a process step that requires organizing data or information according to a standard format.
	Sort	Indicates the sorting of data, information, materials into some pre-defined order.

Table 1.6 Data Processing Symbols in Flowchart

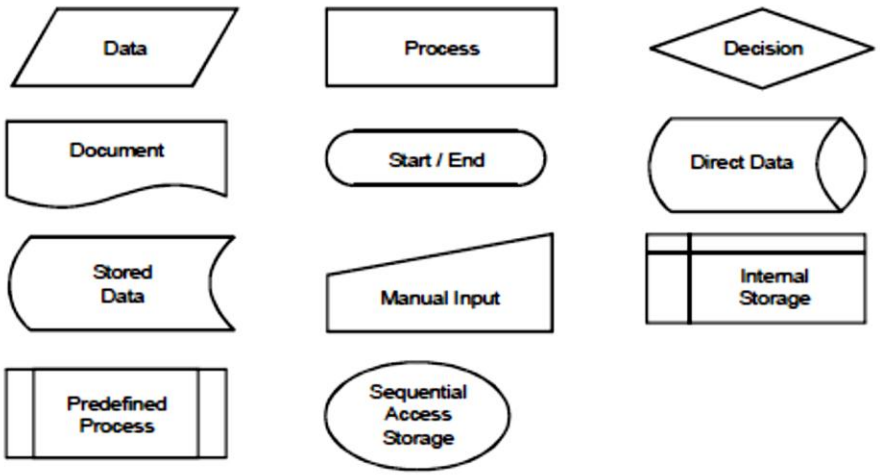


Figure 1.7 Commonly used Flowchart Symbols

3. Guidelines for drawing Flowcharts

The following are some guidelines in flowcharting:

- All necessary requirements should be listed out in logical order for drawing a proper flowchart.
- The flowchart should be clear, neat and easy to follow.
- The usual direction of the flow of a process is from left to right or top to bottom.
- Only one flow line should come out from a process symbol.

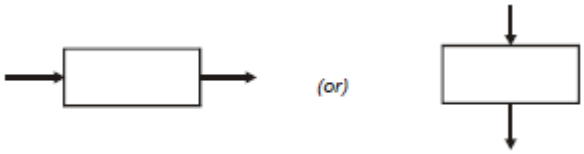


Figure 1.8 (a) One flow line in process

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

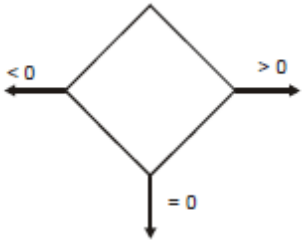


Figure 1.8 (b) Multiple flow lines in a decision

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

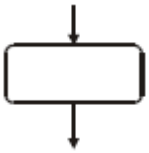


Figure 1.8 (c) One flow line in terminal

- If the flowchart becomes complex, it is better to use connector symbols to reduce the number of flow lines.
- Ensure that the flowchart has a logical start and finish.
- It is useful to test the validity of the flowchart by passing through it with a simple test data.

4. Advantages of using Flowcharts

Communication

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

Effective analysis

- With the help of flowchart, problem can be analysed in more effective way.

Proper documentation

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

Efficient Coding

- Flowcharts act as a guide or blueprint during the systems analysis and program development phase.

Proper Debugging

- Flowchart helps in debugging process.

Efficient Program Maintenance

- The maintenance of running program becomes easy with the help of flowchart.

5. Limitations of using Flowcharts

Complex logic

- Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.

Alterations and Modifications

- If alterations are required the flowchart may require re-drawing completely.

Reproduction

- As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.

1.4.3 Programming Language

Computers are machines. These can work only when instructed. Computers can never think and communicate like human beings. Programming languages bridge the human beings and computers. A programming language is a notation for writing programs, which are specifications of a computation or algorithm.A programming language may also describe computation on some, possibly abstract, machine. It is generally accepted that a complete specification for a programming language includes a description, possibly idealized, of a machine or processor for that language. The following are the qualities of a programming language;

- Languages are not designed to provide a means for having a two-way dialog witha computer.
- It is a set of instructions specified by the human on what thecomputer should do.
- Provides a way for humans to communicate to computers
- It always follows grammar; hence there is no ambiguity

- A system will be able to infer the computer language

Any language has grammar. Similarly a computer programming language also has set of valid words and grammatical rules for instructing a computer to perform specific tasks. The term programming language usually refers to high level languages, such as BASIC, C, C++, COBOL, FORTRAN, Ada, and Pascal.

Advantages of using Algorithm

- Algorithm is a step-by-step representation of a solution to a given problem, which is very easy to understand.
- It has got a definite procedure.
- It is easy for the programmer to first develop an algorithm, then flowchart, and then the computer program
- Algorithm is independent of programming languages.
- It is easy to debug as every step is got its own logical sequence.

Limitations of using Algorithm

- It is time consuming and cumbersome as an algorithm is developed first, which is then converted to flowchart and then converted to computer program.

1.5 STEPS IN PROBLEM SOLVING

The problem solving process starts with the problem specification and ends with a correct program. The steps in the problem solving process are:

1. Problem Definition
2. Problem Analysis
3. Algorithm Development
4. Program Coding
5. Program Testing and Debugging
6. Documentation

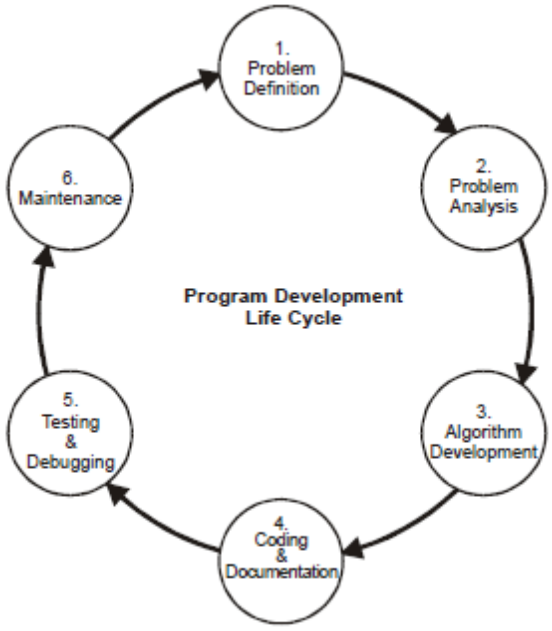


Figure 1.9 Steps in problem solving

1. Problem Definition

Define or specify the problem by answering to the following questions:

- What the computer program do?
- What tasks will it perform?
- What kind of data will it use, and where will it get the data?
- What will be the output of the program?
- How will the program interact with the computer user?

Specifying the problem requirements forces the programmer to state the problem clearly and gain a clear understanding of what is the solution.

2. Problem Analysis

Analysing the problem involves

- Identifying the problem inputs
- Check any additional requirements or constraints
- Determine the required format of the results to be displayed

3. Algorithm Development

Find an algorithm for its solution. An algorithm is a finite set of steps defining the solution of a particular problem. Write step-by-step procedure and then verify that the algorithm solves the problem as intended. The algorithm development can be expressed by

- Pseudo code
- Pseudo code describes the outline of a program, written in a form that can easily be converted into real programming statements.
- Flowchart
- Flowchart is the diagrammatic representation of the algorithms

4. Program Coding

Coding or programming is the process of translating the algorithm into the syntax of a given programming language. Convert each step in the algorithm into one or more statements in a programming language.

5. Program Testing and Debugging

Program testing means running the program and executing all its instructions or functions step by step. Debugging is the process of finding and correcting program code mistakes. Errors in the program may be

- Syntax errors
- Run-time errors
- Logic errors (or so called bugs)

Sl.No.	***Testing***	***Debugging***
1	Finding and locating the defect.	Fixing that defect.
2	Done by the Testing team.	Done by the Development team.
3	Intention behind is to find as many defect as possible.	Intention is to remove those defects.

6. Documentation

Documenting the program is done by

Internal documentation

- Internal documentation consists of remarks or comments written with the program instructions to explain what is being done in the program.

External documentation

- External documentation is made up of the manuals or help menus written about the solution.

1.6 RECURSION AND ITERATION

1.6.1 Iteration

Iterative programs are programs that follow a path from the starting instruction till the end of the algorithm. In an iterative program loop invariant specifies that path which has to be repeated to achieve the desired end result. The measure of progress specifies how far the current step is from the starting step or how near it is from the last step. The basic structure of an Iterative program is

begin routine
*_pre-cond_*
codepre-loop% Establish loop invariant
loop
_loop-invariant _
*exit when _exit-cond_*
codeloop%Make progress while maintaining the loop invariant
end loop
codepost-loop% Clean up loose ends
*_post-cond_*
end routine

Steps to develop an Iterative Program

- **Define Problem :** The problem that needs an iteration has to be identified and defined.
- **Initial Conditions:** The condition that has to satisfied to start the iteration.
- **Define Loop Invariants:** The variable that controls the number of iteration has to be defined.
- **Define Step:** The steps that are to be repeated must be defined.
- **Define Measure of Progress:** The loop invariants that have been defined would be changed when the algorithm progresses. How it progresses has to be defined.
- **Define Exit Condition:** When the iteration should be stopped has to be indentified.
- **Make Progress:** Move forward after executing an instruction.
- **Maintain Loop Invariants:** In order to repeat the steps the loop invariant must be maintained in a range.
- **Ending:** When the iteration has to stop.

Problem: Givenaset of numbers, find the maximum

Input: List of numbers

Output: Maximum Number
Step 1: Begin
Step 2: i = 1; j = 1
Step 3: Begin Loop
Step 3.1 _loop-invariant_: N[i] ismax in N[1.. j].
Step 3.2 exit when (j= n)
Step 3.3 j = j + 1//Make progress while maintaining
the loop invariant
Step 3.4 if(N[i] <N[j]) then i = j
Step 4: End loop
Step 5: Return(i)
Step 6: End

1. Types of Iteration

There are different types of iterative algorithms depending on how the measure of progress and loop invariants is used. The following are the types.

i) Depending on Output

The measure of progress is amount of output constructed and loop invariant is the output constructed so far. The example for this type is selection sort. The given list of array elements has to be sorted. At each iteration the smallest element from the unordered list is chosen and placed in the ordered list. Here the measure of progress is number of elements sorted. The loop invariant is list of sorted and unsorted elements.

PROGRAM 11

Algorithm for the selection sort is given below.

Algorithm: Selection Sort

Step 1: Assume element in index 0 as MIN
Step 2: Compare MIN will all the other elements in the list
Step 3: If an element lesser than MIN exits; place it in index 0
Step 4: Now assume element in index 1 as MIN; repeat steps 2 & 3
Step 5: Repeat until list is sorted

ii) Depending on Input

The iteration depends on the input. The measure of progress is the amount of input considered and loop invariant is amount of process completed on the input. The example for this type is insertion sort; where the measure of progress is the number k of elements inserted and the loop invariant states that the k inserted elements are sorted within a list and that, as before, the remaining elements are off to the unsorted side.

Algorithm: Insertion Sort

Step 1: Compare the first and second element; arrange in order
Step 2: Pick third element; Compare with elements before; arrange in order
Step 3: Pick next element; Compare with all elements before; arrange in order
Step 4: Repeat until till the last element; until the list is sorted

iii) Optimized Search

If you are searching for an element; the search space can be narrowed to give an efficient and optimized searching result. Here the measure of progress is narrowed search space. The loop invariant is element in the narrowed space. The example for this type is binary search.

Algorithm: Binary Search

Step 1: Let lower be 1 and upper be number of elements ; n
Step 2: Set upperBound = number of elements ; n
Step 3: if upper < lower
    return FindElement does not exists.
Step 4: Let mid = lower + (upper - lower) / 2
Step 5: if Element in mid position lesser than FindElement
    set lower = mid + 1
Step 6: if Element in mid position greater than FindElement
    set upper = mid - 1
Step 7: if Element in mid position equal to FindElement
    return FindElementfound in location mid
Step 8: Repeats steps 3 to 7 until FindElement is found

Specialized Measure of Progress

In some iterative programs the measure of progress is some creative function. For example in bubble sort; the number of pairs that are not ordered will be the measure of progress. The algorithm for bubble sort is

Algorithm: Bubble Sort

Step 1: For all the elements in the list
Step 1.1:Compare the adjacent elements; swap if not in order

1.6.2 Recursion

Iterative algorithms start at the beginning and take one step at a time towards the final destination. Another technique is to break the given task into a number of subtasks, solve each of it separately, and then combine the results into one result for the original task. It is called divide-and-conquer method. When the subtasks are of different nature, several functions are written; however when the subtasks are similar it leads to recursive algorithms. The iterative algorithm works in forward fashion. It executes a step and moves to the next; the loop variant shows the path next to be taken. However a recursion algorithm works backward. The last subtask is first completed and the algorithm progresses to the first backwards to find the solution. The advantages of recursive algorithms over iterative are one; sometimes it is easier to work backward than forward and two; it is allowed to have more than one subtask to be solved.

The structure of a recursive algorithm;

algorithmrecursive(A)
_ pre-cond_:A- arguments to the algorithm
_ post-cond_:B - Last subtask reached
begin
if A reached the last subtask B; complete that task and start moving backwards

else
breakA into smaller task C
Now invoke recursive(C)
end algorithm

In a recursive algorithm; the algorithm calls itself with "smaller (or simpler)" input values. When a problem can be solved by using the solutions of smaller versions of the same problem then the recursive algorithm is used to solve that problem. The recursive programs require more memory and computation compared with iterative algorithms, but they are simpler and a natural way of solving a problem.

In a recursive algorithm the condition is verified during the execution of the algorithm and not at the end as in iterative algorithms. The recursion is called again if the end condition is not satisfied. When the condition is fulfilled, the execution of the last called program is resumed exactly to the point in which it called itself. This happens for all the calls of the recursive algorithm; in reverse order.

Problem: Given a non-negative integer n, compute Fn.

Definition:F0 = 0, F1 = 1, Fi = Fi-1 + Fi-2 for i 2.
Input: A nonnegative integer n.
Output: The Fibonacci number Fn

```
Integer FibRec( Integer n)
if ( n <=1 )
        return n ;
else
        return FibRec (n-1) + FibRec (n-2);
```

Problem: Given non-negative integers a and b, not both 0, compute gcd(a,b)

Definition:Fora,b □ 0,gcd(a,b)=
a if b =0,
gcd(b, a mod b) otherwise.
Input :Non negative integers a and b, not both zero.
Output :The greatest common divisor of a and b.

```
Integer gcd(Integer a, Integer b)
if (b == 0)
        return a;
else
        return gcd( b, a mod b);
```

Problem: Given two numbers k and n; find n^k

Input: n,k.
Output: kth power of n

Algorithm: Power_of_N(k)

Step 1 : ifk = 0, then return 1 else return 2*Power_of_N(k-1).
The same problem can be solved by an iterative algorithm.
Algorithm Power_of_N(k):

Step 1 : Set power =1
Step 2 : Set i = 0
Step 3 : if (i <k) calculate power := power * N
Step 4 : i := i + 1
Step 5 : Repeat steps 3 and 4
Step 6 : Return power

Problem statement: Add 10 and 20

Problem description: To solve this problem, take a variable sum and set it to zero. Then, take two numbers 10 and 20 as input. Next, add both the numbers and save the result in the variable sum i.e., sum = 10 + 20. Finally, print the value stored in the variable sum.

Algorithm

Step 1: Initialize sum = 0
Step 2: Enter the numbers 10 and 20.
Step 3: Add them and store the result in sum
Step 4: Print sum

Flowchart

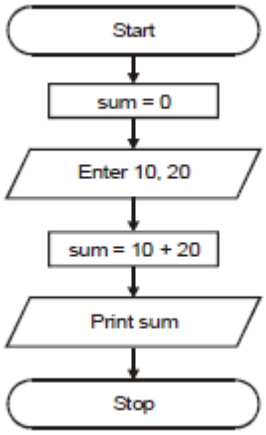


Figure 1.11 Addition

Problem statement: Display area and circumference of the circle

Problem description:
Get the radius of the circle as the input from the user.
Find the area and circumference of the circle, using the following formulas:
area = π * r2
circumference = 2 * π * r
π value is unknown to the computer. Hence, initialize the value for as 3.14 in the computer program.

Algorithm

Step 1: Start
Step 2: Read radius and store to r
Step 3: Assign value of π to P
Step 4: Find area and circumference
area = π * r2

circumference = 2 * π * r

Step 5: Display area and circumference

Step 6: Stop

Flowchart

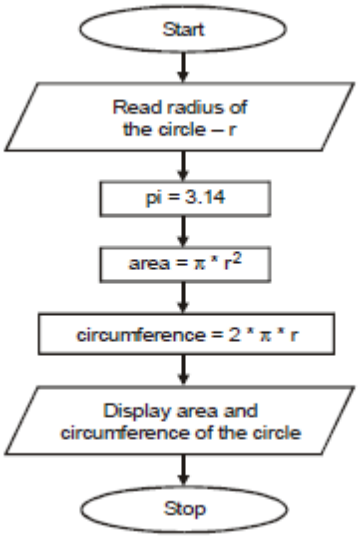


Figure 1.12 Area and Circumference

Problem statement: Find the sum of 5 numbers.

Problem description:

The question is to find the sum of 5 numbers. Take two variables - sum and count and set both of them to zero. The sum variable will store the result while the count variable will keep track of how many numbers taken. To solve this problem, the concept of loop is used. In loop or iterative operation, execute some steps repeatedly as long as the given condition is TRUE. In this case, it is reading the input ie. reading 5 numbers. Initialize sum and count to zero. Then, read the input and store it in a variable n. Next, add the value stored in n to sum and save the answer in sum.

i.e., sum = sum + n

Then, increment count by 1 and check if count is less than 5. If this condition is TRUE then take another input. If the condition is FALSE then print the value stored in variable sum.

Algorithm

Step 1: Initialize sum = 0 and count = 0

Step 2: Enter n

Step 3: Find sum + n and assign it to sum and then increment count by 1

Step 4: if count < 5

if YES go to step 2

else

Print sum

Flowchart

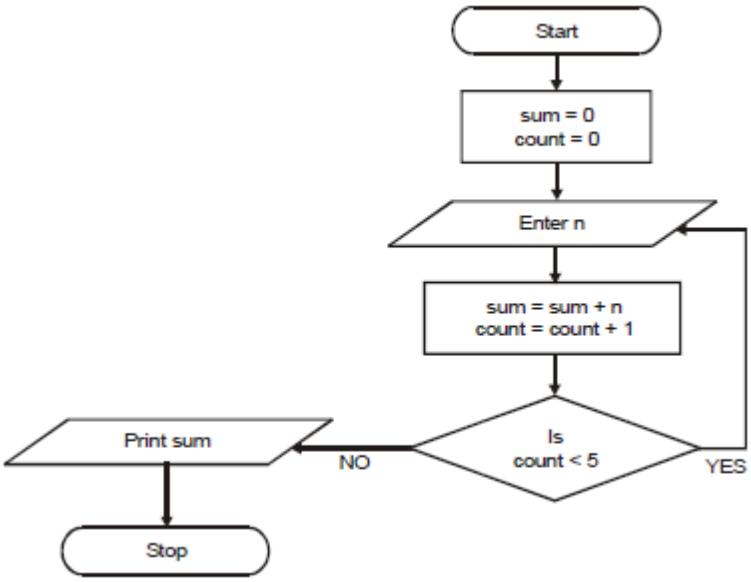


Figure 1.13 Sum

Problem statement: Print Hello World 10 times.

Problem description:

This problem is solved using the loop concept. Take a variable count and set it to zero. Then, print "Hello World" and increment count by 1.

i.e., count = count + 1

Check if count is less than 10. If this is TRUE then print "Hello World" and increment the variable count. If the condition is FALSE then stop.

Algorithm

Step 1: Initialize count = 0
Step 2: Print "Hello World"
Step 3: Increment count by 1
Step 4: if count < 10
if YES go to step 2
else stop

Flowchart

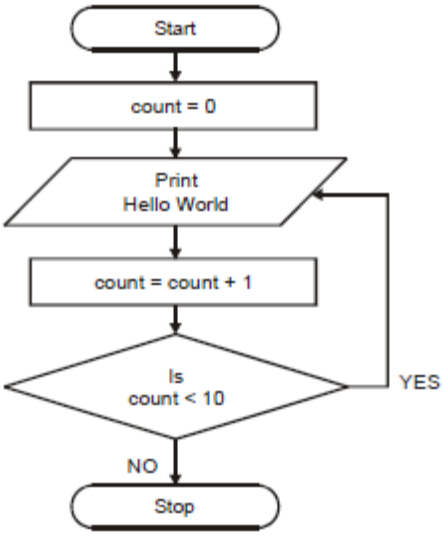


Figure 1.14 Print

Problem statement: Find the sum of the numbers in each set.

Problem description:

Given is the hundred numbers divided in ten sets in the following order.

Set 1: 1-10

Set 2: 11-20

…

Set 10: 91-100

Algorithm

Step 1: Initialize count = 1 and i = 1

Step 2: Check if i is less than or equal to 10

if YES then perform step 3

else STOP

Step 3: Set sum = 0 and j = 1

Step 4: Check if j is less than or equal to 10

if YES then perform step 5

else perform step 9

Step 5: Add count to sum

sum = sum + count

Step 6: Increment count by 1

count = count + 1

Step 7: Increment j by 1

j = j + 1

Step 8: Go to Step 4

Step 9: Print sum

Step 10: Increment i by 1

i = i + 1

step 11: Go to Step 2

Flowchart

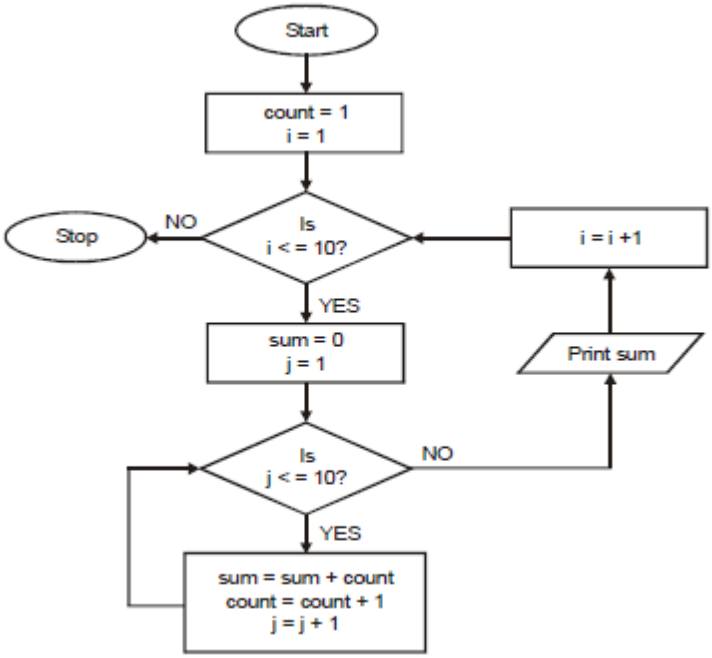


Figure 1.16 Set sum

1.8 ILLUSTRATIVE PROGRAMS

Problem statement : Find minimum in a list

Problem description :

Minimum in a list of elements can be achieved in different ways. One ways is to sort the list of elements in ascending order and get the first element as minimum. Sorting algorithms are explained elaborately in Unit IV. 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 then assume second the minimum and keep repeating the procedure till the last element.

Algorithm

Step 1: Get the list of elements
Step 2: Assume first element as MIN
Step 3: Compare MIN with next element
Step 4: If MIN is greater than next element; set MIN=next element
Step 5: Repeat steps 3 and 4 till the last element
Step 6: Display MIN as the minimum element of the list

Problem statement: Insert a card in a list of sorted cards

Problem description:

In general the sorting is in ascending order and sorted cards would be either playing card with numbers or flash cards with alphabets. In that case, a person would check the new card's number or alphabets with the set of cards sorted. All the least cards will be picked out and when a card with greater number/alphabet is encountered; the new card is placed above the greater card and the picked out cards will be replaced above the new card. In case of playing cards which have Jack,

Queen and King along with one to ten numbers; the type of card, numbers and alphabets must be considered. The below algorithm considers the playing cards.

Algorithm

Step 1: Get the sorted cards
Step 2: Get card to be inserted - Insert_Card
Step 3: Consider first card for comparison; Current_Card
Step 4: Compare Current_Card type with Insert_Card
Step 5: If type is not same;
Step 5.1: pick out 13 cards
Step 5.2: Consider the first card of new as Current_Card
Step 6: If type is same
Step 6.1: If Insert_Card is a number card compare the numbers in the list and place in the right order
Step 6.2: If Insert_Card is a role card compare JACK, QUEEN and KING order and place in the right order

Problem statement: Tower of Hanoi, consists of three towers and more than one rings of different sizes as shown in the below picture.

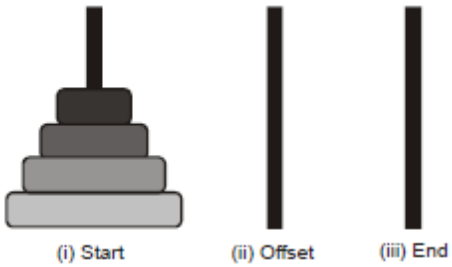


Figure 1.17 Towers of Hanoi

In the beginning the rings are stacked upon in an ascending order. The problem is to move all the disks to another tower; always maintaining the order. A few rules to be followed for Tower of Hanoi are -

- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.
- No large disk can sit over a small disk.

Problem description:
The easiest way to solve this problem is to find solution
for the least number of disks; that is two. The three towers
are named as start, end and offset. If we have 2 disks -

- First, we move the smaller (top) disk to offset tower.
- Then, we move the larger (bottom) disk to end tower.
- And finally, we move the smaller disk from offset tower to end tower

Now consider more number of disks. In that case we can divide the stack of disks in two parts. The largest disk (nth disk) is in one part and all other (n-1) disks are in the second part. Now move disk n from start to end and then put all other (n1) disks onto it. The best way to do is using recursion.

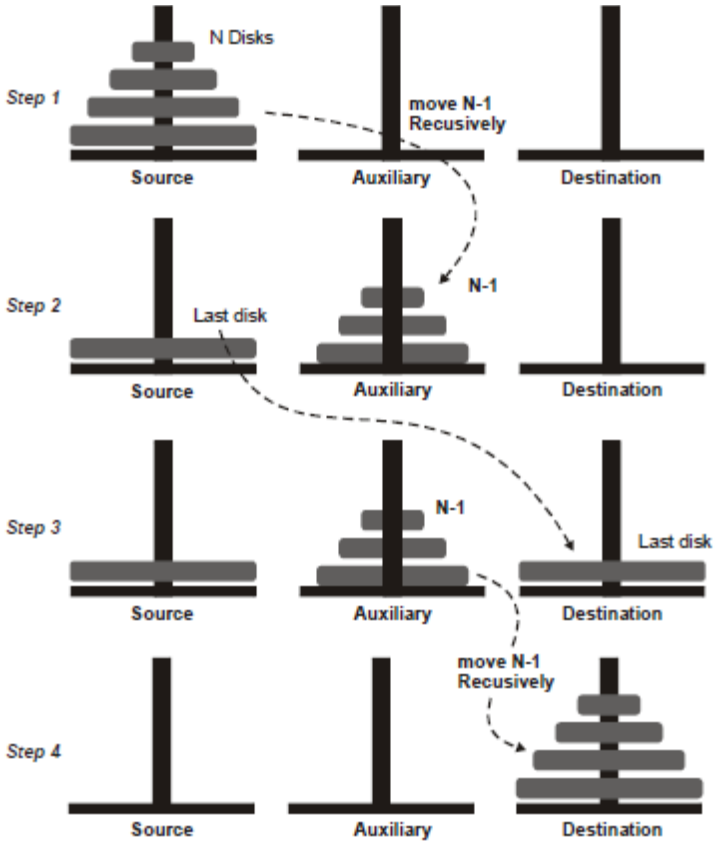


Figure 1.18 Moves of Discs

Algorithm

TowersOfHanoi(disk, start, end, offset)

Step 1: If only one disk is there then move disk from start to end

Step 2 : If more disks are there then

Step 2.1: Recursively call TowersOfHanoi(disk-1, start, offset, end)

Step 2.2: Move disk from start to end

Step 2.3: Recursively call TowersOfHanoi(disk-1, offset, end, start)

Problem statement:

The number-guessing game is played by two players. Player A thinks of a number within the given range of numbers; say 10 to 100. The Player B has to find out the number correctly. The Player A would be giving the number of digits in the number he thought of.

Problem description:

The Player B can start guessing from the starting number of the range till he reaches the correct number. This approach is called linear search, because he guesses all the numbers in a sequence. But it takes several guesses and hence is not efficient. If Player A can give the Player B clue on whether the number he as guessed is lesser or greater than what he thought of, it would be reduce the range of number into half. Keep going, always eliminating half of the remaining numbers. This approach is called binary search. For a range of 1 to 30, Player B can find the number in at most 5 guesses with this technique.

Algorithm

Step 1: Get the range - Start and End
Step 2: Get the guess number G and thought number T
Step 3: If G is lesser than the T then set Start as G
Step 4: If G is greater than the T then set End as G
Step 5: Repeat steps 3 and 4 until G and T are the same TCS Codevita Problems

TWO MARKS QUESTIONS WITH ANSWERS

1. Define Computer.

A computer is an electronic data processing device, which accepts and stores data input, processes the data input, and generates the output in a required format.

2. Define algorithm.

The sequence of steps to be performed in order to solve a problem by the computer isknown as an algorithm.

Programs = Algorithms + Data

3. What are the two phases in algorithmic problem solving?

The algorithmic problem solving actually comes in two phases:

- Derivation of an algorithm that solves the problem
- Conversion of the algorithm into program code

The first phase is the algorithmic phase and the second phase is the coding phase.

4. Why Algorithmic phase is the difficult phase? Justify

The algorithmic phase is the difficult phase, for two main reasons.

- Firstly, it challenges the mental facilities to search for the right solution.
- Secondly, it requires the ability to articulate the solution concisely into step by-step instructions, a skill that is acquired only through lots of practice.

5. What are the steps involved in algorithm development process?

The algorithm development process consists of five major steps.

Step 1: Obtain a description of the problem.
Step 2: Analyse the problem.
Step 3: Develop a high-level algorithm.
Step 4: Refine the algorithm by adding more detail.
Step 5: Review the algorithm.

5. Compare Computer Hardware and Software.

Description	Hardware	Software
Definition	Devices that are required to store and execute the software.	Computer Software is a set of instructions for a computer to perform specific operations.
Types	Input Devices, Output Devices, Storage Devices, Processing Devices, Control Devices.	System Software, Programming Software, Application Software.

Example	CD-ROM, Keyboard, Monitor, Printer, Scanner.	Adobe Acrobat, Microsoft Word
Nature	Hardware is physical in nature.	Hardware is logical in nature.

7. State some of the properties of an algorithm.

- Finiteness
- Definiteness
- Effectiveness
- Input
- Output
- Feasibility
- Generality

8. What are the three building blocks in a Algorithm?

Building Block	***Common name***
Instruction	Action/Sequence
State/Selection	Decision
Control Flow/Iteration	Repetition or Loop

9. Define assignment, input and output statement.

- **Assignment:** Any instruction that assigns value to a variable
- **Input:** Any instruction that gets the input values
- **Output:** Any instruction that displays or prints the input values

10. Describe different notations in algorithm and classify each.

Algorithms can be expressed in many different notations, including Natural Language, pseudo code, flowcharts and programming languages.

- Natural language tends to be verbose and ambiguous, and is rarely used.
- Pseudo code and flowcharts are structured ways to express algorithms.
- Programming languages are intended for expressing algorithms in a form that can be executed by a computer.

11. Define Pseudocode.

Pseudocode is an informal language used by programmers to develop algorithms. Pseudocode is a "text-based" detail design tool.

Example:

```
Enter value
if value greater than 10
        say “Your number is greater than 10”
else
        say “Your number is less than 10”
```

12. Define some of the rules to be followed on Pseudocode.

Rules to follow:

- Only one statement per line

- Capitalized initial keyword
- Indent to show hierarchy.
- End multi-line structures.
- Keep statement language independent.

13. Define flowchart.

A flowchart is a graphical representation of an algorithm. A flowchart is a diagram made up of boxes, diamonds and other shapes, connected by arrows
- each shape represents a step in the process, and the arrows show the order
in which they occur.

14. What are the different types of flowcharts?

- Document flowcharts
- Data flowcharts
- System flowcharts
- Program flowchart

15. What are the symbols used in flowcharting type?

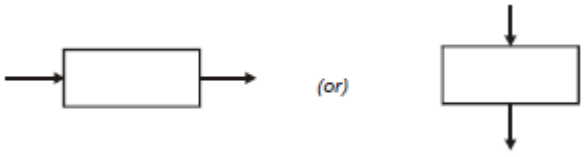
- Process / Operation Symbols
- Branching and Control of Flow Symbols
- Input and Output Symbols
- File and Information Storage Symbols
- Data Processing Symbols

16. Describe the Data Processing Symbols in Flowchart.

- Collate
- Sort

17. List some guidelines for drawing flowchart.

- The flowchart should be clear, neat and easy to follow.
- The usual direction of the flow of a process is from left to right or top to bottom.
- Only one flow line should come out from a process symbol.



- (
- All necessary requirements should be listed out in logical order for drawing a proper flowchart.

18. List the merits in drawing the Flowcharts.

- Easy Communication
- Used for Effective analysis
- Proper documentation
- Efficient Coding
- Proper Debugging

- Efficient Program Maintenance

19. List the demerits in drawing the flowchart.

- Complex logic
- Alterations and Modifications
- Reproduction

20. Explain some of the qualities of Programming Language?

- Languages are not designed to provide a means for having a two-way dialog with a computer.
- It is a set of instructions specified by the human on what the computer should do.
- Provides a way for humans to communicate to computers.
- It always follows grammar; hence there is no ambiguity.
- A system will be able to infer the computer language.

21. What is the difference between the algorithm and the program?

A program is the implementation of an algorithm to be run on a specific computer and operating system. An algorithm is more abstract.

22. Give the Pseudocode to check the biggest of 2 numbers.

```
if (A>B)
        Print "A is greater"
else
        Print "B is greater"
```

23. State the differences between Iteration and Recursion.

Iteration Recursion

Iteration explicitly uses a repetition Recursion achieves repetition through structure. repeated function calls. Iteration keeps modifying the counter Recursion keeps producing simple until the loop condition fails. versions of the original problem until the base case is attained. It reduces the processor's operating It increases the processor's operating time. time. Iteration normally occurs within the Recursion causes another copy of the loop, hence no need of extra memory. function; hence considerable memory space is occupied.