accenture

Capgemini

eClerx

genpact

HCL

Microsoft

ORACLE

pwc

Tech Mahindra

ZOHO

wipro

Unit-I
Introduction to Algorithm & Data Structure

Presented by,
Prof. Barkha M Shahaji
Department of Computer Engineering,
Trinity College of Engineering and research, Pune-48

SYLLABUS

Introduction: From Problem to Program (Problem, Solution, Algorithm, Data Structure and Program).

Data Structures: Data, Information, Knowledge, and Data structure, Abstract Data Types (ADT), Data Structure Classification (Linear and Non-linear, Static and Dynamic, Persistent and Ephemeral data structures)

Algorithms: Problem Solving, Introduction to algorithm, Characteristics of algorithm, Algorithm design tools: Pseudo-code and flowchart.

Complexity of algorithm: Space complexity, Time complexity, Asymptotic notation- Big-O, Theta and Omega, finding complexity using step count method, Analysis of programming constructs-Linear, Quadratic, Cubic, Logarithmic.

Algorithmic Strategies: Introduction to algorithm design strategies- Divide and Conquer, and Greedy strategy.

INTRODUCTION

Introduction

Computer : A programmable device that can store, retrieve, and process data.(Combination of H/w & S/w)

Hardware : things which we can touch.

Software : things which we cannt touch.(Can only see)

Programming: A programming language is a set of commands, instructions, and other syntax use to create a software program.

Data : Information in a form a computer can use

Information : Any knowledge that can be communicated

Introduction

Data type : The specification of how information is represented in the computer as data and the set of operations that can be applied to it Computer

Computer program : Data type specifications and instructions for carrying out operations that are used by a computer to solve a problem

Introduction

Machine language : The language, made up of binarycoded instructions, that is used directly by the computer

Assembly language : A low-level programming language in which a mnemonic is used to represent each of the machine language instructions for a particular computer

Assembler : A program that translates an assembly language program into machine code

Introduction

Compiler : A program that translates a program written in a high-level language into machine code

Source code : Data type specifications and instructions written in a high-level programming language

Object code : A machine language version ofsource code.

DATA STRUCTURE

DATA STRUCTURE

Data Structure is **a way to store and organize data so that it can be used efficiently.**

Data :

"Data is nothing but **collection of information** i.e. facts or figures."

Data Object :

"Data object is a **region of storage** that contains a value or group of value"

NEEDOFDATA STRUCTURE

1. Stores huge data

2. Stores data in systematic way

3. Retains logical relationship

4. Provides various structure

5. Static and dynamic formats

6. Better algorithms

ABSTRACT DATA TYPE

ADT :

"**Abstract data** types are **mathematical models of a set of data values or information** that share similar behavior or qualities and that can be specified and identified independent of specific implementations.

Abstract data types, or ADTs, are typically used in algorithms."

Another definition of ADT is

ADT is set of D, F and A.

D – domain = Data object

F – function = set of operation which cannt carried out on data object.

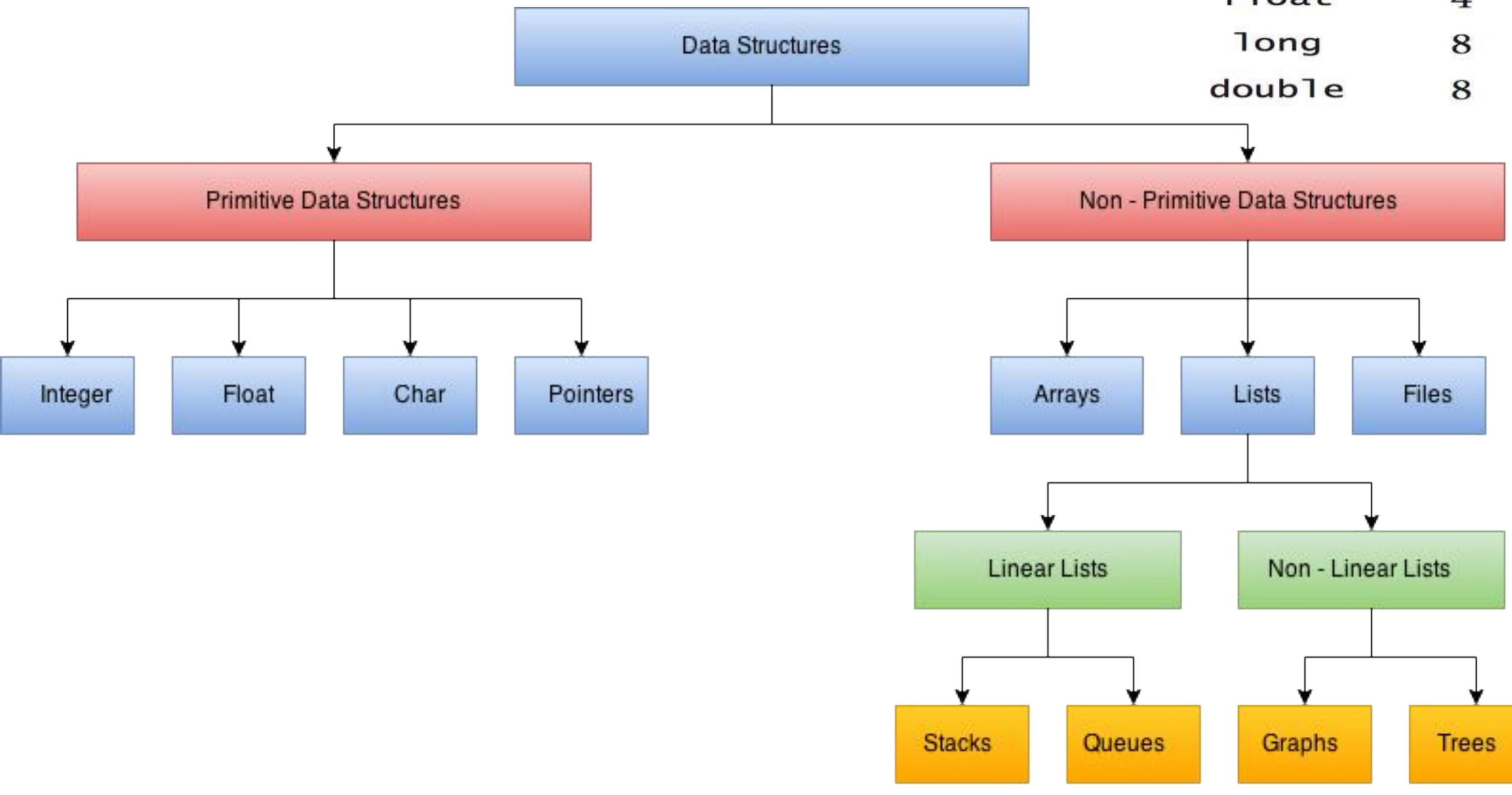
A – axioms = Properties and rule of the operation

TYPES OFDATA STRUCTURE

There are two types :

1. Primitives data structure
2. Non-primitive data structure

type	*bytes*
boolean	1
byte	1
char	2
int	4
float	4
long	8
double	8



TYPES OFDATA STRUCTURE

1. Primitives data structure :

"**Primitive data structures** are those which are predefined way of storing **data** by the system. "

e.g. int, char, float etc

2. Non-primitive data structure :

"The **data** types that are derived from primary **data** types are known as **non-Primitive data** types. These datatype are used to store group of values."

e.g. struct, array, linklist, stack, tree , graph etc.

Linear and Non-Linear Data Structure

1. **Linear Data Strucute** :

"Linear data structure traverses the **data** elements sequentially, in which only one **data** element can directly be reached"

Ex: Arrays, Linked Lists, stack, queue.

2. **Non-Linear Data Strucute :**

"Every **data** item is attached to several other **data** items in a way that is specific for reflecting relationships."

Ex: Graph , Tree

Linear vs Non-Linear Data Structure

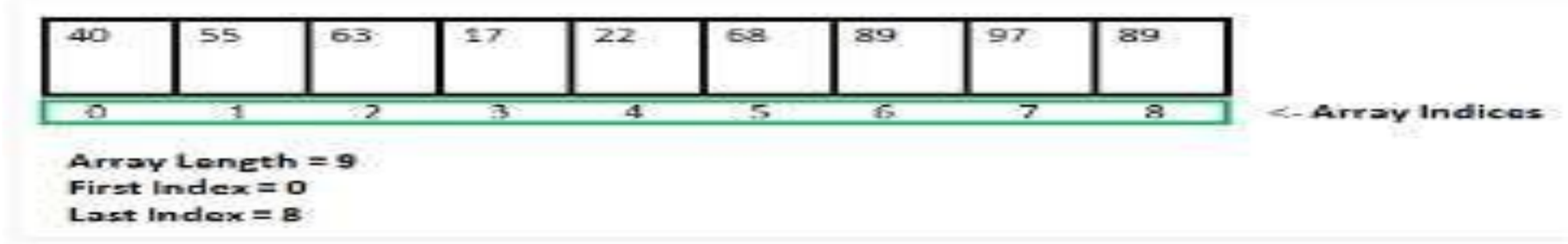
LINEAR DATA STRUCTURES	NON-LINEAR DATA STRUCTURES
Linear Data structures are used to represent **sequential** data.	Non-linear data structures are used to represent **hierarchical** data.
Linear data structures are **easy** to implement	These data structures are **difficult** to implement.
Implementation: Linear data structures are implemented using array and linked lists	**Implementation:** Non-linear data structures are mostly implemented using linked lists.
e.g: The basic linear data structures are list, **stack and queue**.	**e.g:** The basic non-linear data structures are **trees and graphs**.
For the implementation of linear data structures, we don't need **non-linear** data structures.	For the implementation of non-linear data structures, we need **linear** data structures.
USE: These are mostly used in application software development.	**USE:** These are used for the development of game theory, artificial intelligence, image processing

Static and Dynamic Data Structure

1. Static data strucure :

"A **static data structure** is an organization or collection of **data** in memory that is fixed in size."

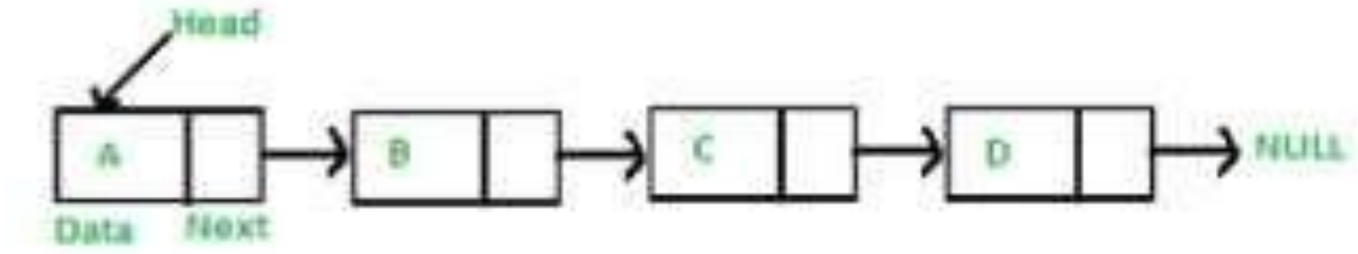
Ex: Arrays



2. Dynamic Data Strucute :

" In **Dynamic data structure** the size of the **structure** in not fixed and can be modified during the operations performed on it"

Ex: Linked list



Persistent and Ephemeral DataStructure

1. Persistent data strucure :

"A **persistent data structure** is a **data structure** that always preserves the previous version of itself when it is modified.."

Ex: Linked list, tree

2. Ephemeral Data Strucute :

" An ephemeral data structure is one of which only one version is available at a time(it does not preserve previous version)."

Ex: RAM , Cache memory

Relationship among Data, Data Structure and Algorithms

Data is considered as set of facts and figures or data is value of group of value which is in particular format.

Data structure is method of gathering as well as organizing data in such manner that several operation can be performed

Problem is definedas a situation or condition which need to solve to achieve the goals

Algorithm is set of ordered instruction which are written in simple english language.

ALGORITHM

ALGORITHM – PROBLEM SOLVING

COMPUTER :

"Computer is multi purpose Electronic Machine which is used for storing , organizing and processing data by set of program

Problem :

"Problem is defined as situation or condition which needs to solve to achive goal"

Steps in Problem Solving :

1. Define the problem
2. Data gathering
3. Decide effective solution
4. Implement and evaluate the solution
5. Review the result.

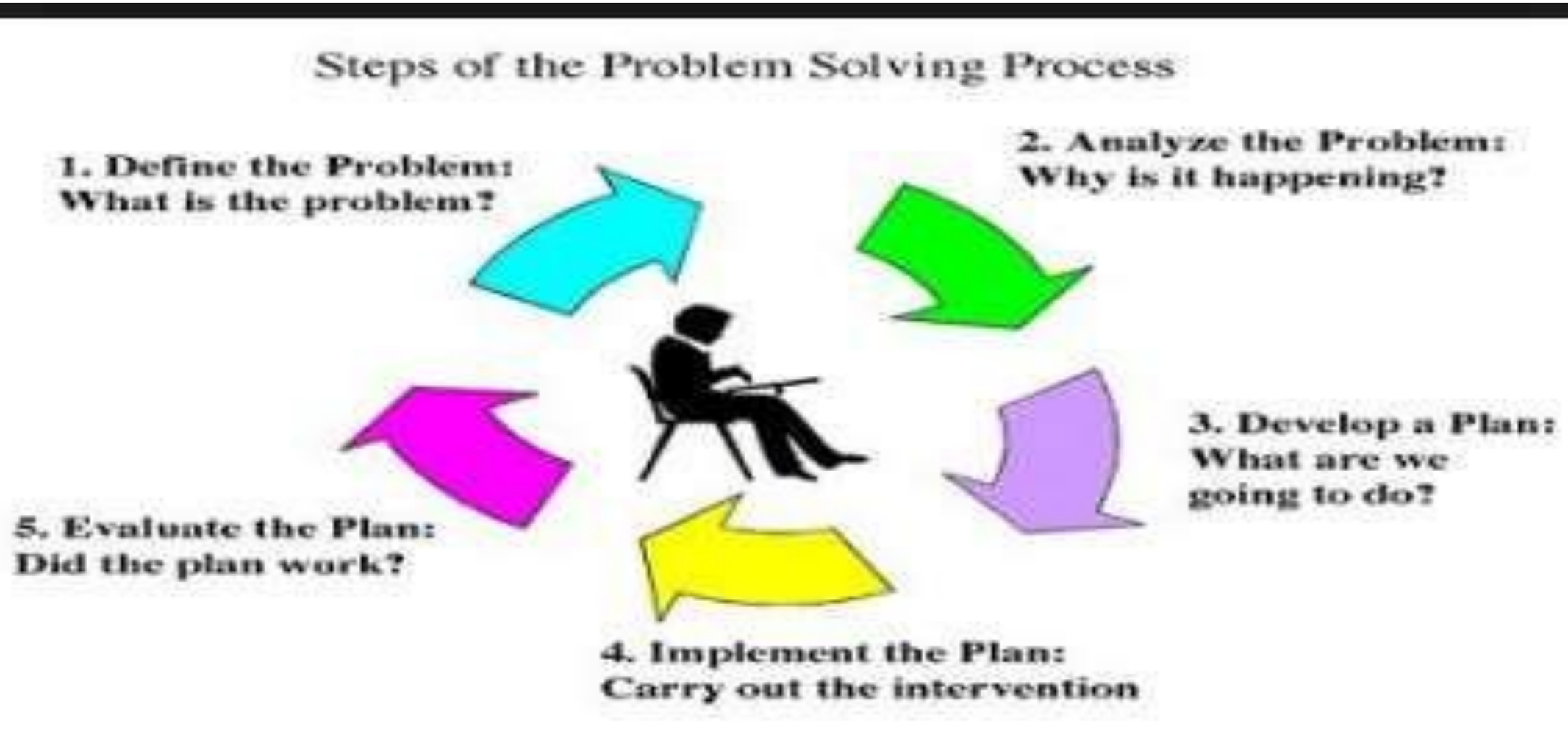
PROBLEM SOLVING TECHNIQUES

There are two types :

1. Algorithmic
2. Flowchart

Algorithms is set of instructions which are writeen in simple english language.

Flowchart is graphical representation of the algorithms.



Some other Problem Solving Techniques

1. Trial and error techniques

2. Divide and conquer techniques

3. Merging solution

4. The building block approach

5. Brain storming techniques

INTRODUCTION OF ALGORITHMS

DEFINITION *:*

"An **algorithm** is defined as a step-by-step procedure or method for solving a problem by a computer in a finite number of steps."

From the data structure point of view, following are **some important categories of algorithms −**

Search − Algorithm to search an item in a data structure.

Sort − Algorithm to sort items in a certain order.

Insert − Algorithm to insert item in a data structure.

Update − Algorithm to update an existing item in a data structure.

Delete − Algorithm to delete an existing item from a data structure.

CHARACTRISTICS OF ALGORITHM

.Unambiguous − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.

2. Input − An algorithm should have 0 or more well-defined inputs.
.Output − An algorithm should have 1 or more well-defined outputs, and should match the desired output.

4. Finiteness − Algorithms must terminate after a finite number of steps.

5. Feasibility − Should be feasible with the available resources.
.Independent − An algorithm should have step-by-step directions, which should be independent of any programming code.

EXAMPLE OF ALGORITHM

Example

Let's try to learn algorithm-writing by using an example.

Problem − Design an algorithm to add two numbers and display the result.

Step 1 − *START*
Step 2 − *declare three integers* ***a, b*** *&* ***c***

Step 3 − *define values of* ***a*** *&* ***b***

Step 4 − *add values of* ***a*** *&* ***b***
Step 5 − *store output of step 4 to* ***c***

Step 6 − *print* ***c***

ALGORITHM DESIGN TOOL

- **There can be two tools :**

1. Flowchart
2. Pseudo Code

Flowchart :

" Flowchart is graphical representation of the algorithms"

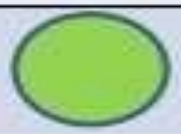
Pseudo Code :

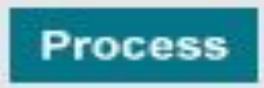
"It is simply an implementation of an algorithm in the form of annotations and informative text written in plain English.

FLOWCHART

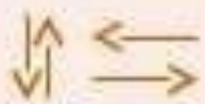
Symbol used in Flowchart :

Flowchart Symbols

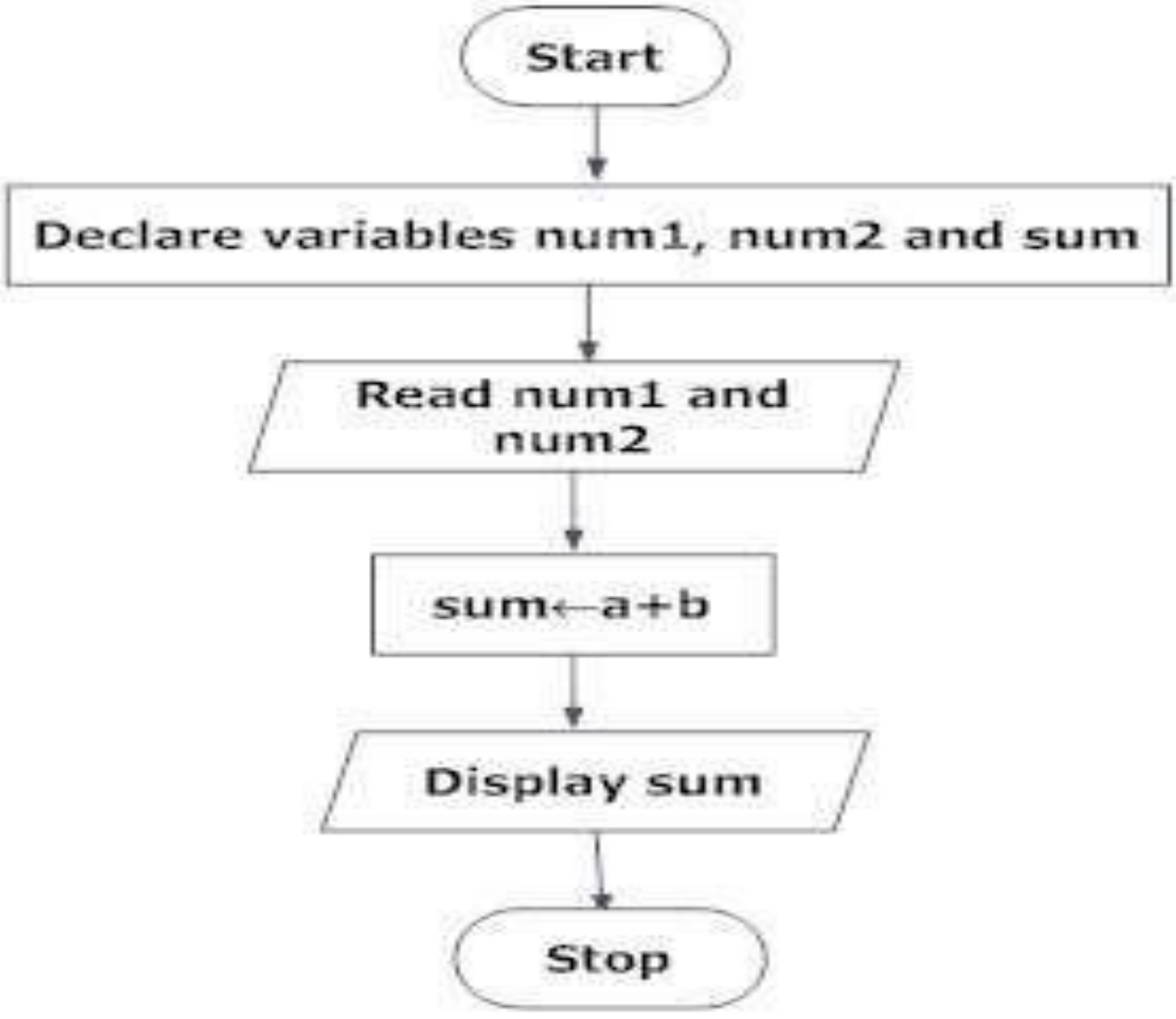
SYMBOL / SHAPE	NAME	FUNCTION
	Terminator	-show the start and stop points in a process
	Decision	-used when there are 2 or 3 options (Yes/No)
	Connector	-show a jump from one point in the process flow to another
	Data (Input/Output)	-indicates inputs and outputs from a process
	Preparation	-set-up operation
	Process	-represents a process, action, or function
	Manual Input	-represents the manual input of data into a computer(keyboard)
	Flow line	-show the direction that the process flows

Name	Symbol	Usage
Start or Stop		The beginning and end points in the sequence.
Process		An instruction or a command.
Decision		A decision, either yes or no.
Input or Output		An input is data received by a computer. An output is a signal or data sent from a computer.
Connector		A jump from one point in the sequence to another.
Direction of flow		Connects the symbols. The arrow shows the direction of flow of instructions.

Symbols Used in Flowcharts

Picture	Shape	Name	Action Represented
	Oval	Terminal Symbol	Represents start and end of the Program
	Parallelogram	Input/Output	Indicates input and output
	Rectangle	Process	This represents processing of action. Example, mathematical operator
	Diamond	Decision	Since computer only answer the question yes/no, this is used to represent logical test for the program
	Hexagon	Initialization/ Preparation	This is used to prepare memory for repetition of an action
	Arrow Lines & Arrow Heads	Direction	This shows the flow of the program
		Annotation	This is used to describe action or variables
	Circle	On page connector	This is used to show connector or part of program to another part.
	Pentagon	Off-page connector	This is used to connect part of a program to another part on other page or paper

EXAMPLE OF FLOWCHART



EXAMPLE FOR ALGORITH & FLOWCHART

Step1: Start

Step2: Initialize the count variable to zero

Step3: Initialize the sum variable to zero

Step4: Read a number say x

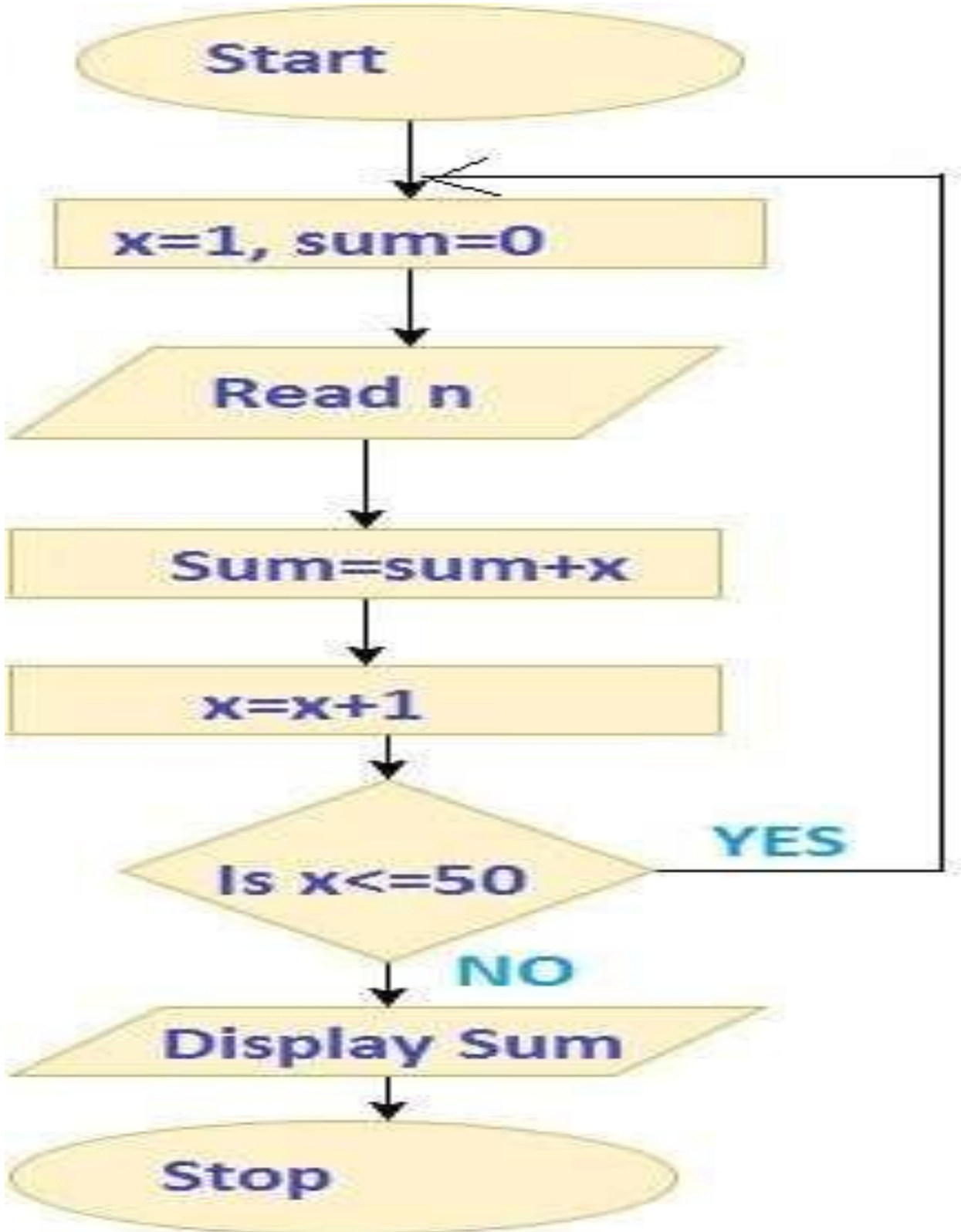
Step 5: Add 1 to the number in the count variable

Step6: Add the number x to the sum variable.

Step7: Is the count variable in the memory greater than 50? If yes, display the sum: go to step 8. No, If Repeat from step 4

Step8: Stop

Design an algorithm and flowchart to input fifty numbers and calculate their sum.



WRITE A PROGRAM FOR ADDING 10 NUMBERS

Write an algorithm for adding 10 numbers.

Solution :

Algorithm as a series as steps

Step 1 : Assign 10 to N

Step 2 : Assign 0 to SUM

Step 3 : Assign 1 to I

Step 4 : if (I > N) go to step 9

Step 5 : read X

Step 6 : Assign SUM + X To SUM

Step 7 : Assign I + 1 to I

Step 8 : Go to step 4

Step 9 : Print SUM

Step 10 : Stop

Flow chart

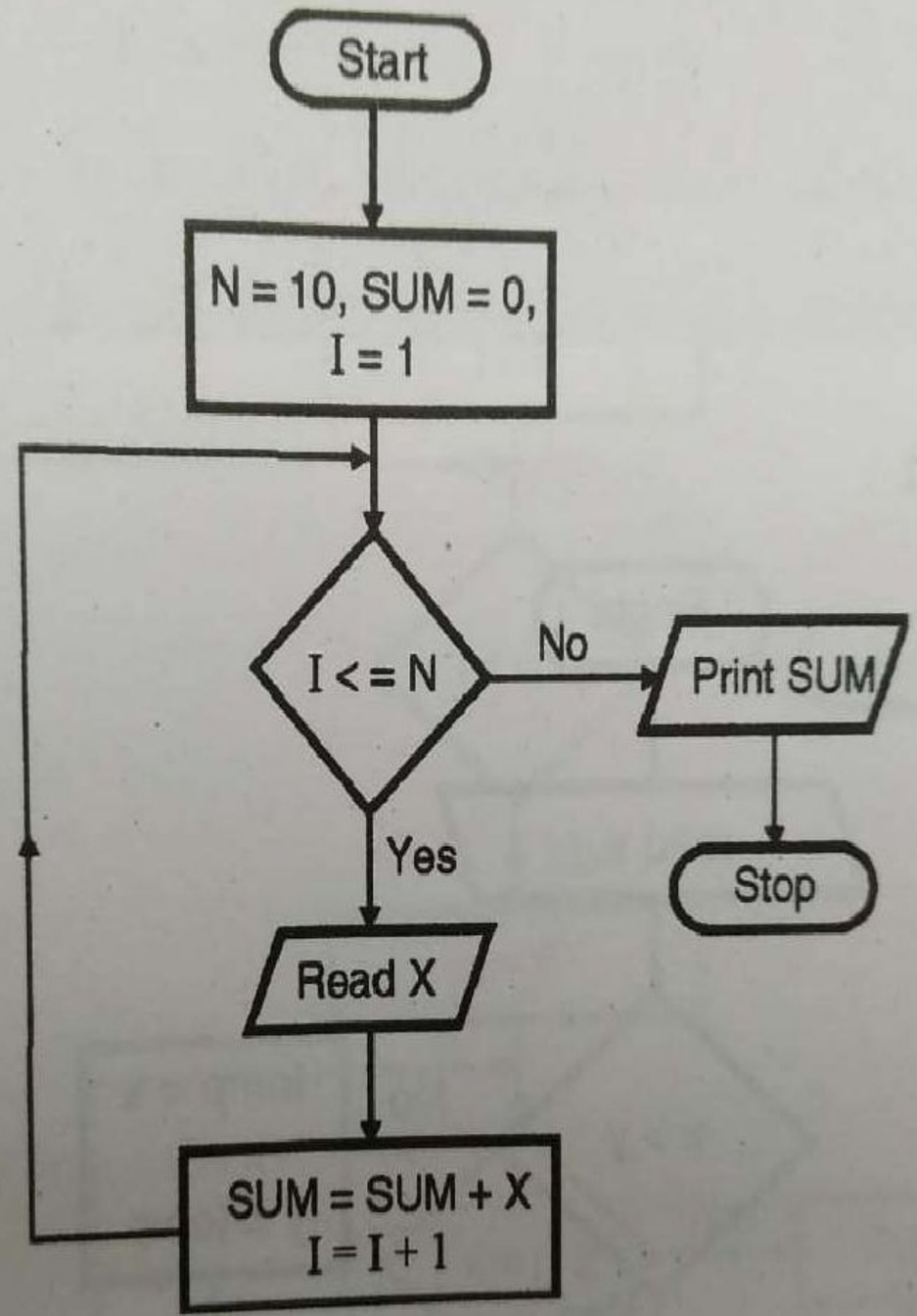


Fig. Ex. 1.1.1 : Flowchart for adding 10 numbers

WRITE A PROGRAM TO FIND FACTORIAL OF NUMBER

Algorithm as a series of steps

Step 1 : read n

Step 2 : Assign 1 to fact

Step 3 : if n < = 0 then go to step 7

Step 4 : Assign n*fact to fact

Step 5 : Assign n-1 to n

Step 6 : goto step 3

Step 7 : Print fact

Step 8 : Stop

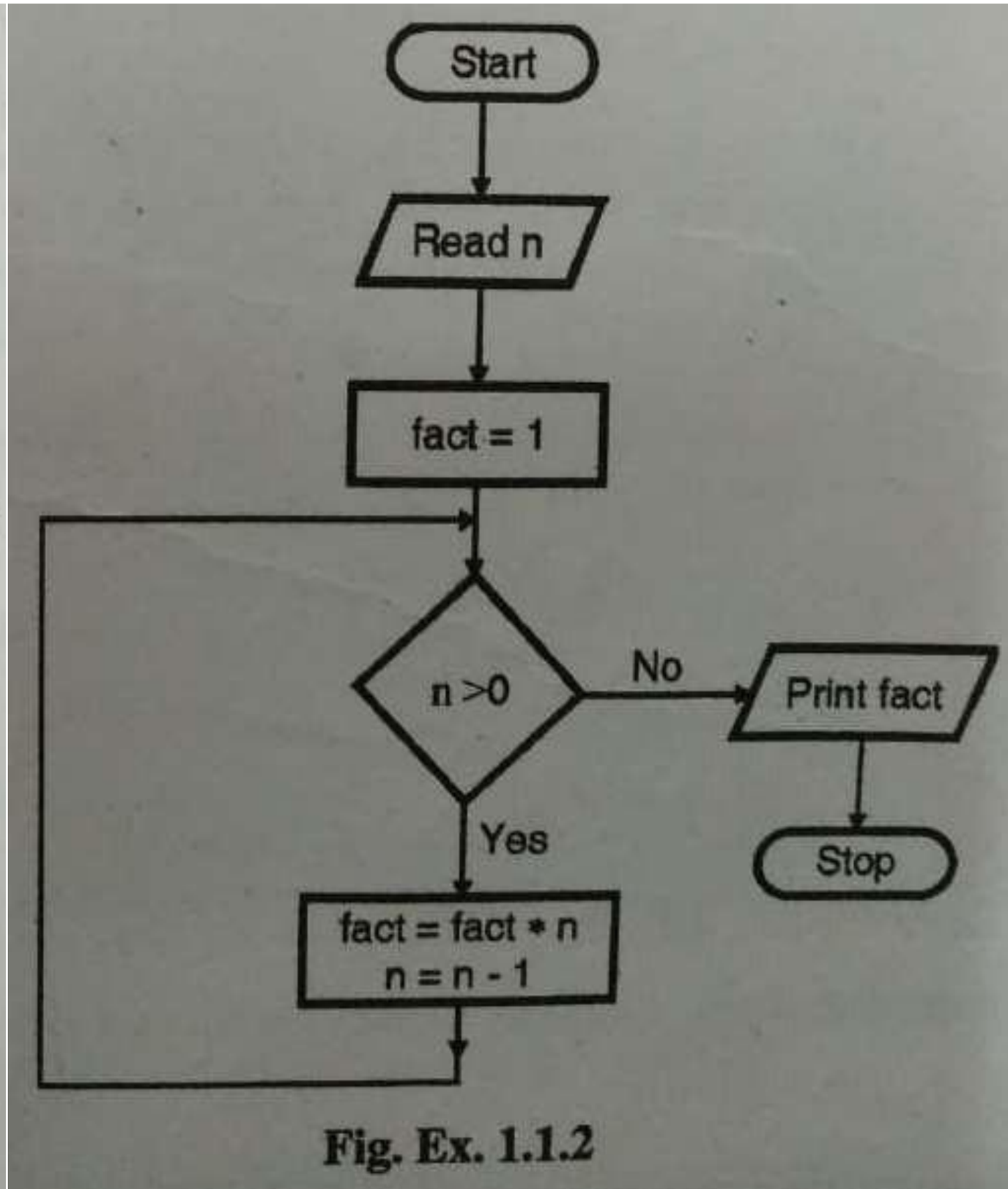


Fig. Ex. 1.1.2

DIFFERENT APPROCHES TO DESIGN ALGORITHMS

Types of approach :

1. Top down approach
2. Bottom up approach

TOP DOWN APPROACH	BOTTOM UP APPROACH
1. Larger problem divided into smaller	Smaller pieces are combined together
2. Execution Start from top to down	Execution start from bottom to top
3. C is top down approach language	C++ is bottom up approach language
4. Main() is written at beginning	Main() is written at end of program

Complexity of Algorithms

ALGORITHM ANALYSIS

- ***A Priori* Analysis** − This is a **theoretical analysis** of an algorithm. Efficiency of an algorithm is measured by assuming that all other factors, for example, **processor speed, are constant** and have no effect on the implementation.

- ***A Posterior* Analysis** This is an **empirical analysis** of an algorithm. The selected algorithm is implemented using **programming language**. This is then executed on target computer machine. In this analysis, actual statistics like running time and space required, are collected.

CASES OF ANALYSIS ALGORITHMS

There are 3 types

1. Worst case
2. Best case
3. Average case

Worst-Case Analysis

- Interested in the worst-case behaviour.
- A determination of the maximum amount of time that an algorithm requires to solve problems of size *n*

Best-Case Analysis

- Interested in the best-case behaviour
- Not useful

Average-Case Analysis

- A determination of the average amount of time that an algorithm requires to solve problems of size *n*
- Have to know the probability distribution
- The hardest

Best Case − Minimum time required for program execution.

Average Case − Average time required for program execution.

Worst Case − Maximum time required for program execution

Standard measure of efficiency

There are two important complexity measures:

1. Time complexity
2. Space complexity

Time complexity :

"The time which is required for analysis of given problem of particular size is known as time complexity"

Space complexity :

"The amount of computer memory required tp solve the given problem of particular size is called as space complexity"

Time efficiency - a measure of amount of time for an algorithm to execute.

Space efficiency - a measure of the amount of memory needed for an algorithm to execute.

Q. 1.6.5 Write an example of Step count. *(Refer section 1.6.2)* **(4 Marks)**

```
sum( a[], 5)
{
        sum =0;
        for(i=0;i<=5;i++)
        {
                sum = sum + a[i];
        }
        return sum;
}
```

Table 1.6.1 : Step Count

Instruction	Step Count
Algorithm sum(a[ ], n)	0
{	0
sum = 0;	1
for(i=0;i<=n;i+)	n + 1
{	0
sum = sum + a[i]	N
}	0
return sum;	1
}	0
Total Steps = 1 + n + 1 + n + 1 = 2n + 3	

Asymptotic notations

Asymptotic Notations are languages that allow us to analyze an algorithm's running time by identifying its behavior as the input size for the algorithm increases. This is also known as an algorithm's growth rate

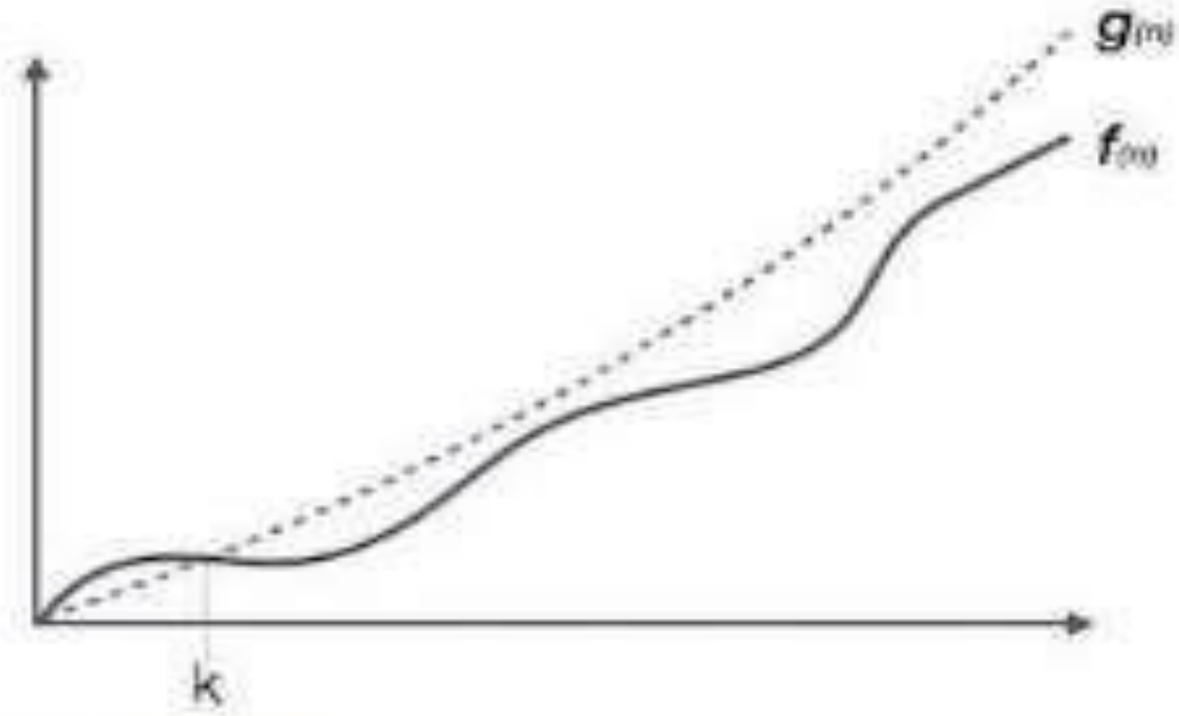
Asymptotic Notation gives us the ability toanswer these questions.

Following are the commonly used **asymptotic notations** to calculate the running time complexity of an algorithm.

1. O Notation
2. Ω Notation

Big Oh Notation, O

The notation O(n) is the formal way to express the upper bound of an algorithm's running time. **It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.**



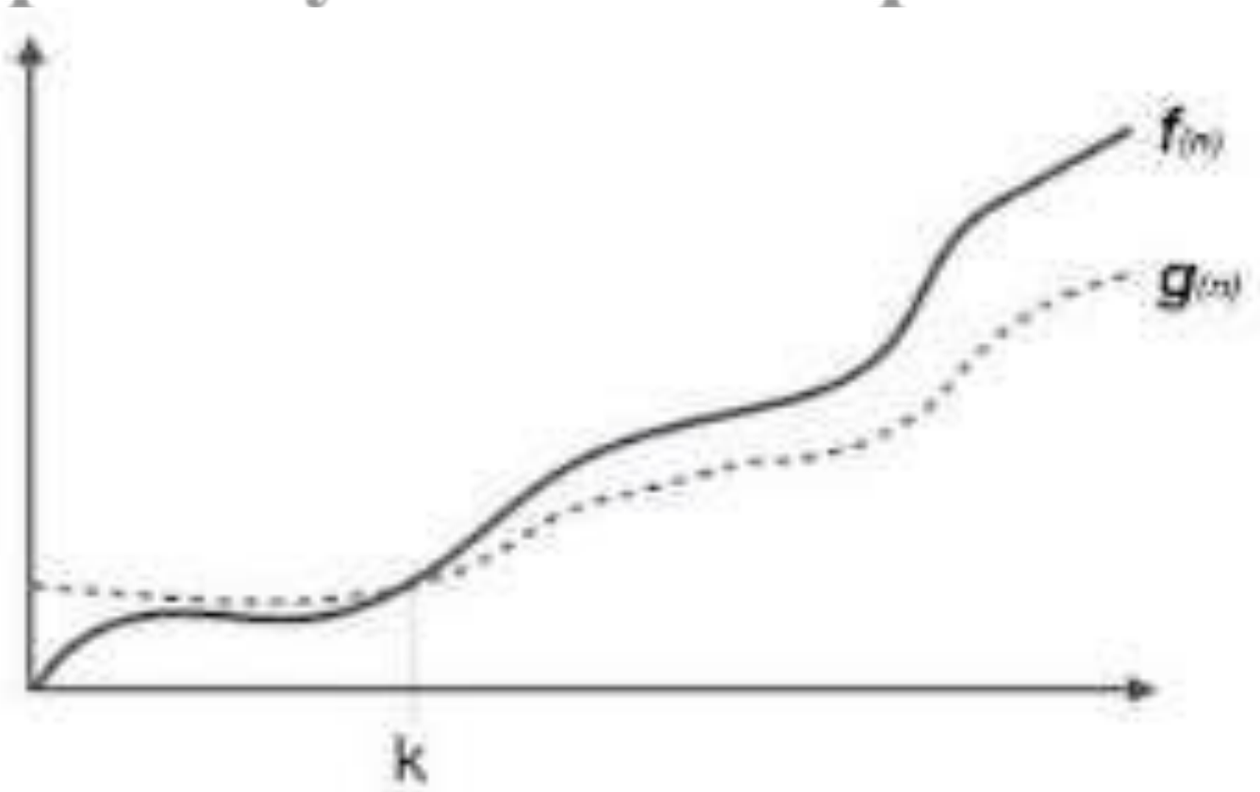
For example, for a function ***f*(n)**

```
O(f(n)) = { g(n) : there exists c > 0 and n0 such that f(n) ≤ c.g(n) for all n > n0. }
```

Omega NOTATION

Omega Notation, Ω

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. **It measures the best case time complexity or the best amount of time an algorithm can possibly take to comp**



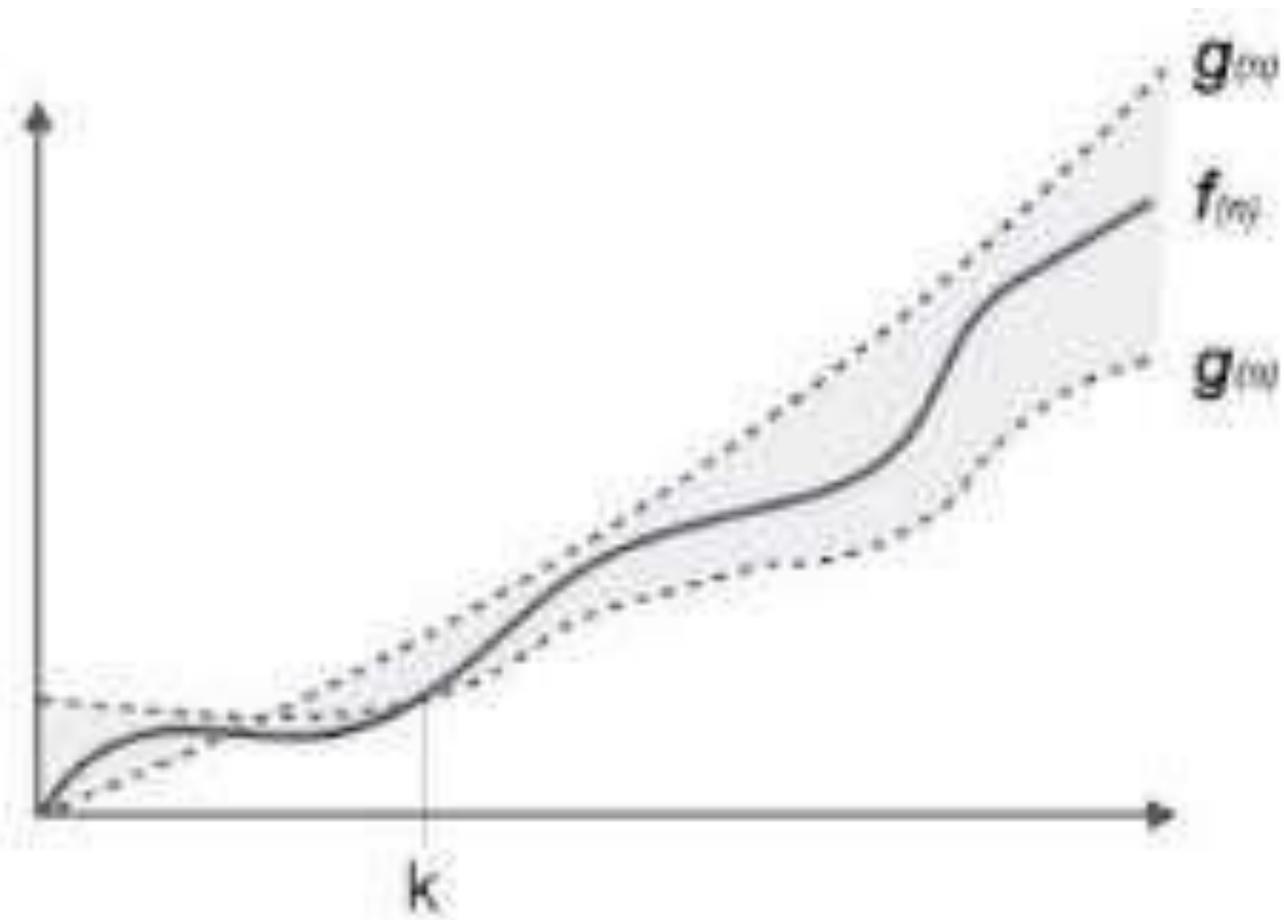
For example, for a function ***f*(n)**

$\Omega(f(n)) \geq \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } g(n) \leq c.f(n) \text{ for all } n > n_0. \}$

Theta NOTATION

Theta Notation, θ

The notation θ(n) is the **formal way to express both the lower bound and the upper bound** of an algorithm's running time. It is represented as follows −



θ(f(n)) = { g(n) if and only if g(n) = O(f(n)) and g(n) = Ω(f(n)) for all n > n0. }

Common Asymptotic Notation

Following is a list of some common asymptotic notations –

constant	–	O(1)
logarithmic	–	O(log n)
linear	–	O(n)
n log n	–	O(n log n)
quadratic	–	$O(n^2)$
cubic	–	$O(n^3)$
polynomial	–	$n^{O(1)}$
exponential	–	$2^{O(n)}$

Analysis of Programming Construct

description	*order of growth*	*example*	*framework*
constant	1	`count++;`	*statement (increment an integer)*
logarithmic	$\log n$	`for (int i = n; i > 0; i /= 2)` `  count++;`	*divide in half (bits in binary representation)*

linear	n	`for (int i = 0; i < n; i++)`
		`  if (a[i] == 0)`
		`  count++;`
			single loop (check each element)

linearithmic	$n \log n$	[*see mergesort* (PROGRAM 4.2.6)]	*divide-and-conquer (mergesort)*

quadratic	n^2	`for (int i = 0; i < n; i++)`
		`  for (int j = i+1; j < n; j++)`
		`  if (a[i] + a[j] == 0)`
		`  count++;`
			double nested loop (check all pairs)

cubic	n^3	`for (int i = 0; i < n; i++)`
		`  for (int j = i+1; j < n; j++)`
		`  for (int k = j+1; k < n; k++)`
		`  if (a[i] + a[j] + a[k] == 0)`
		`  count++;`
			triple nested loop (check all triples)

exponential	2^n	[*see Gray code* (PROGRAM 2.3.3)]	*exhaustive search (check all subsets)*

Algorithmic Strategies

ALGORITHMIC STRATEGIES

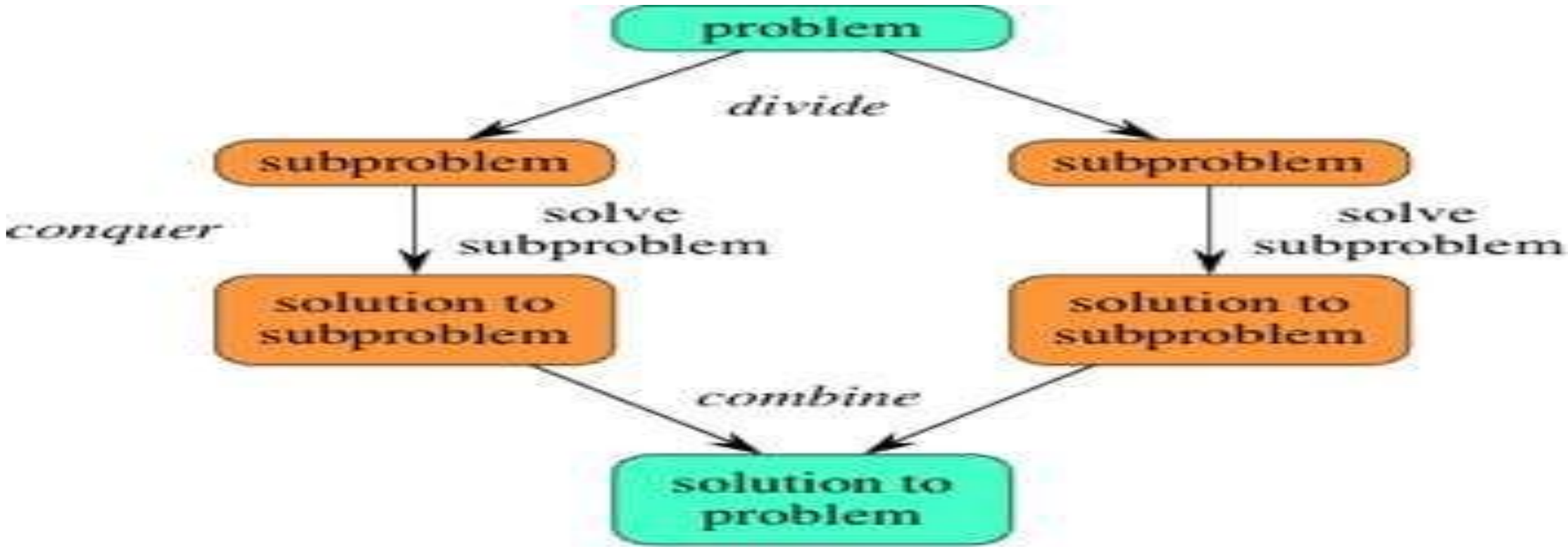
Algorithm design strategies are the general approaches used to develop efficient solution to problem.

Algorithm Strategies are :

1. Divide and conquer
2. Merge sort
3. Recursive algorithm
4. Backtracking algorithms
5. Heuristic algorithms
6. Dynamic Programming algorithm

DIVIDE AND CONQUER

In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the subproblems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. Those "atomic" smallest possible sub-problem (fractions) are solved. The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.



DIVIDE AND CONQUER

Operation for strategy :

Divide – Break the problem into subproblem of same type

Conquer – Recursively solve these sub problem

Combine – Combine the solution of sub problem

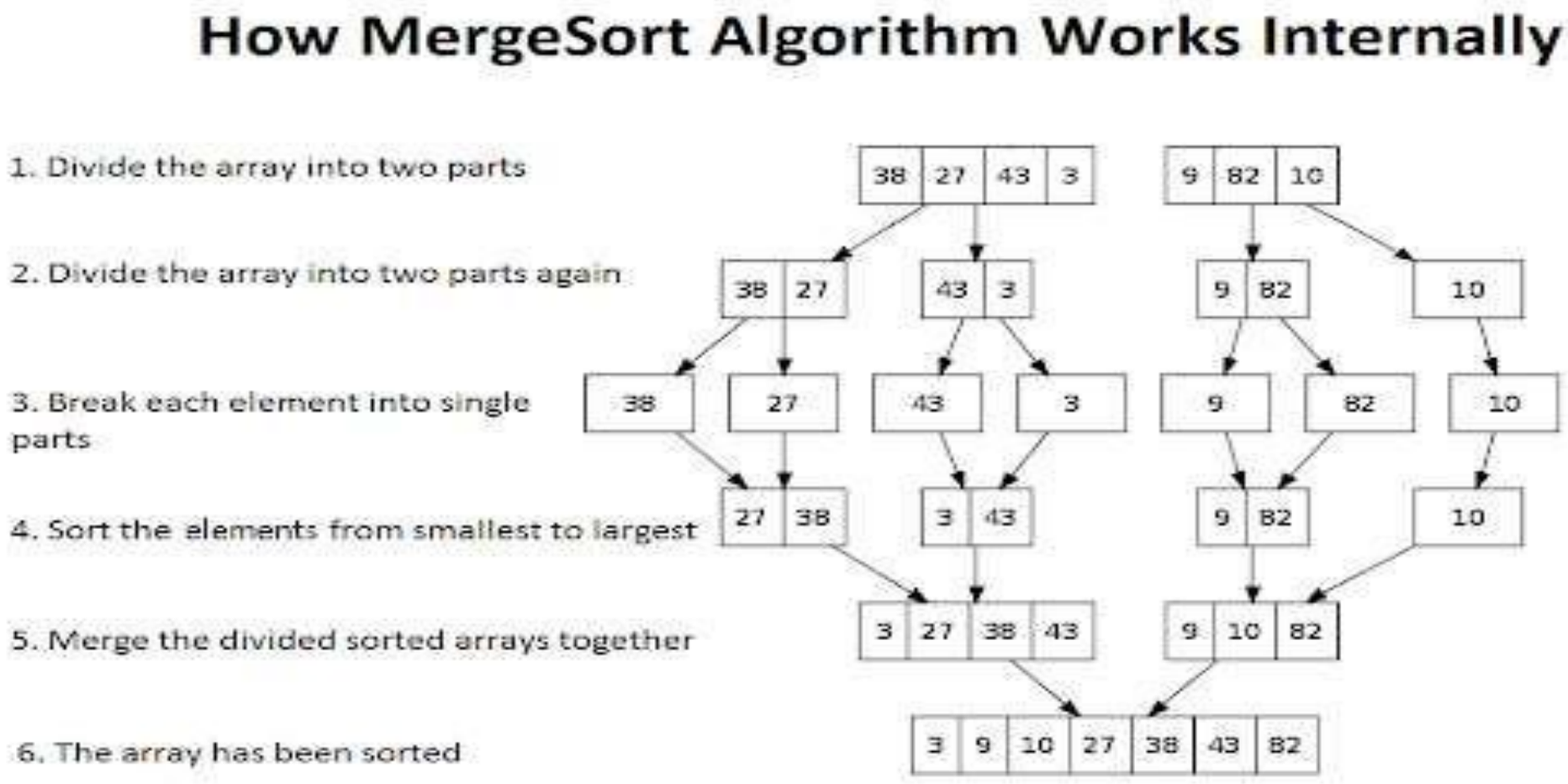
Following **algorithms are based on divide and conquer** strategies :

1. Merge sort
2. Binary search
3. Quick sort
4. Closest pair
5. Tower of Hanoi

DIVIDE AND CONQUER

1. Merge sort :

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.



DIVIDE AND CONQUER

2. Tower of Hanoi :

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following **simple rules:**

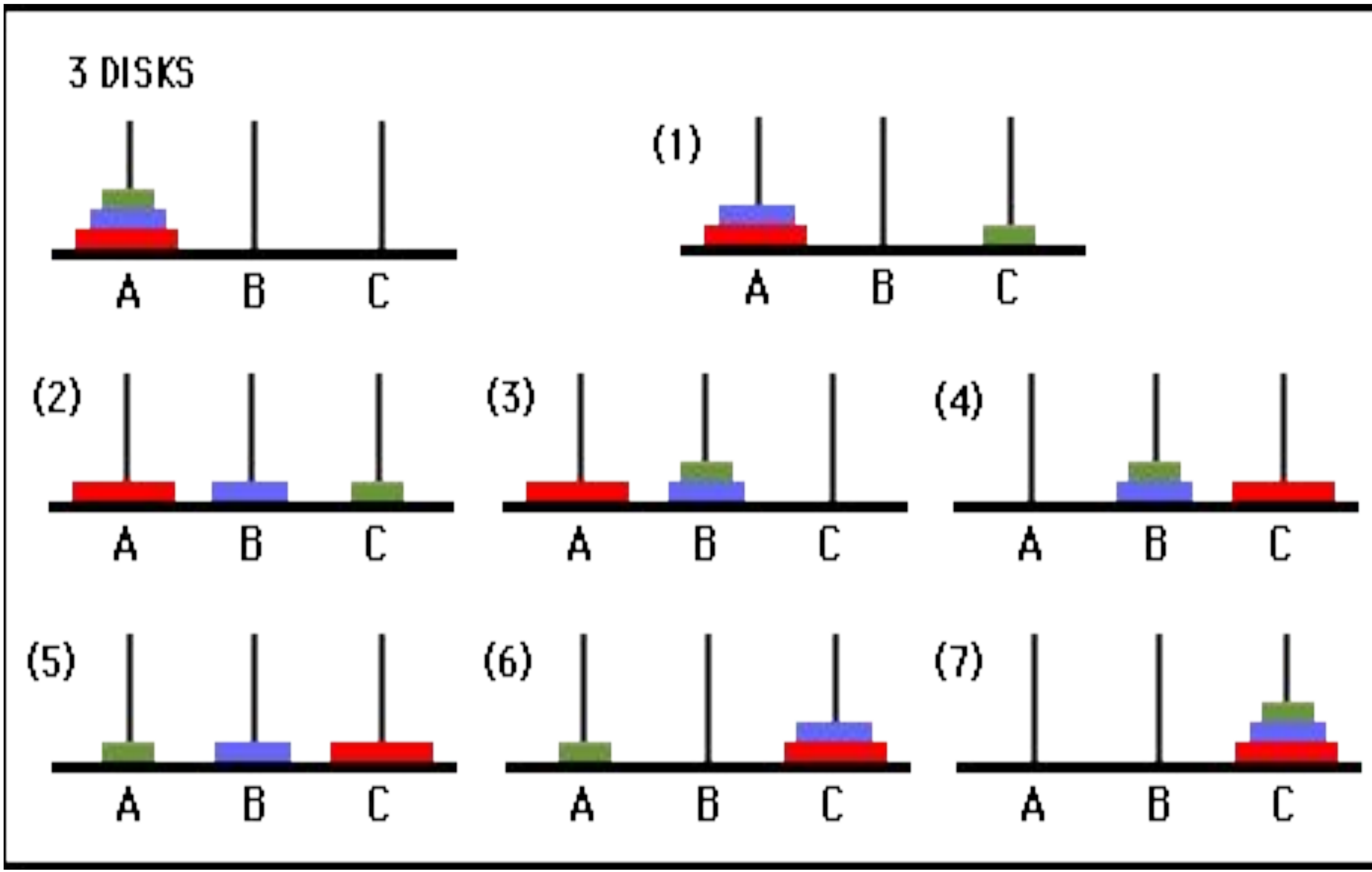
simple rules

1) Only one disk can be moved at a time.
2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

3) No disk may be placed on top of a smaller disk.

DIVIDE AND CONQUER

2. Tower of Hanoi : Example



GREEDY STRATEGIES

Greedy algorithm :

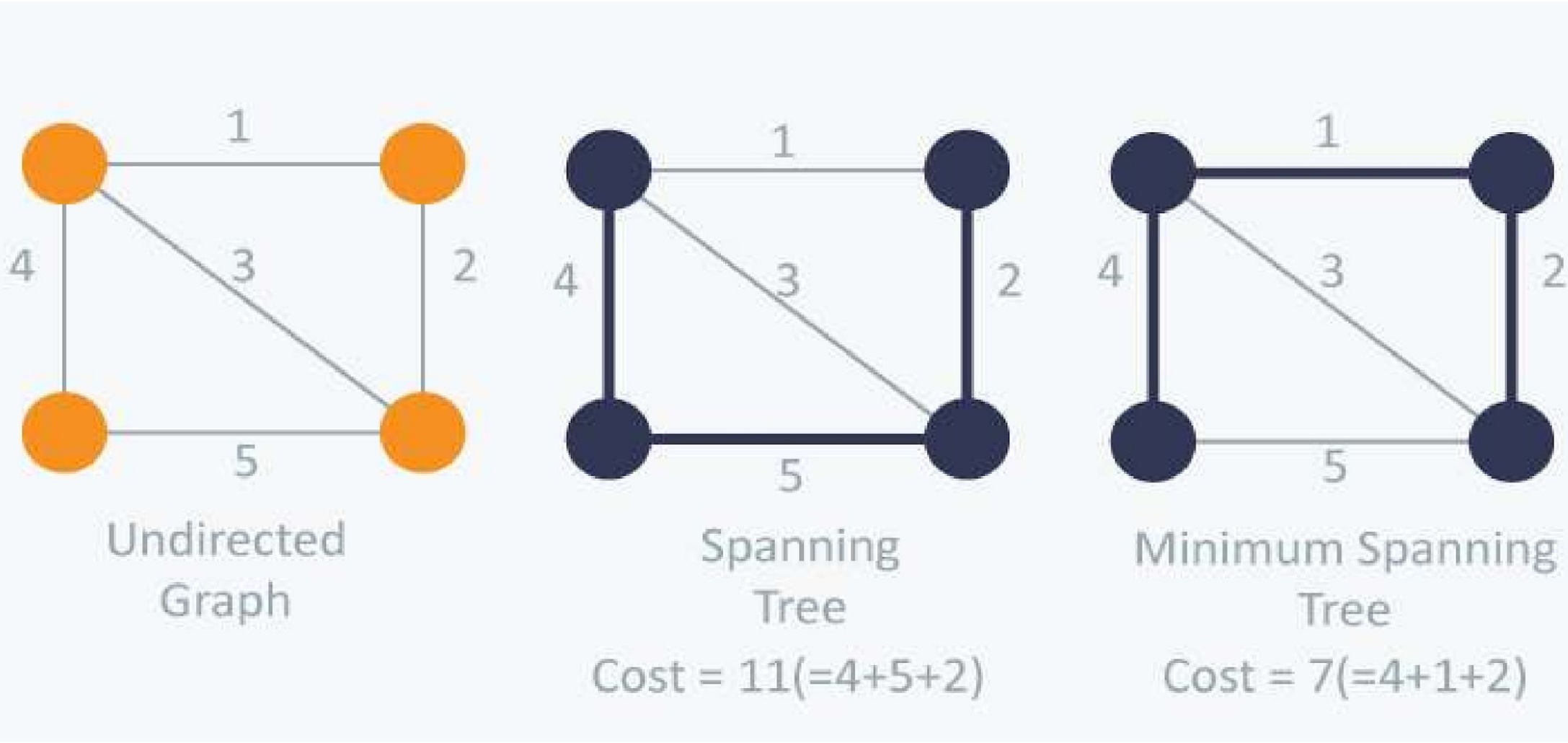
An algorithm is designed to achieve **optimum solution** for a given problem. In greedy algorithm approach, decisions are made from the given solution domain. As being greedy, **the closest solution that seems to provide an optimum solution is chosen.**

Example of greedy strategy :

1. Travelling Salesman Problem
2. Prim's Minimal Spanning Tree Algorithm
3. Kruskal's Minimal Spanning Tree Algorithm
4. Dijkstra's Minimal Spanning Tree Algorithm
5. Knapsack Problem
6. Job Scheduling Problem

1. Minimum Spanning tree (Prims or Kruskal's algorithms)

The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees. Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees. There also can be many minimum spanning trees.



GREEDY STRATEGIES

1. **Kruskal's algorithms :**

Kruskal's Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge which has least weight and add it to the growing spanning tree.

Algorithm Steps :

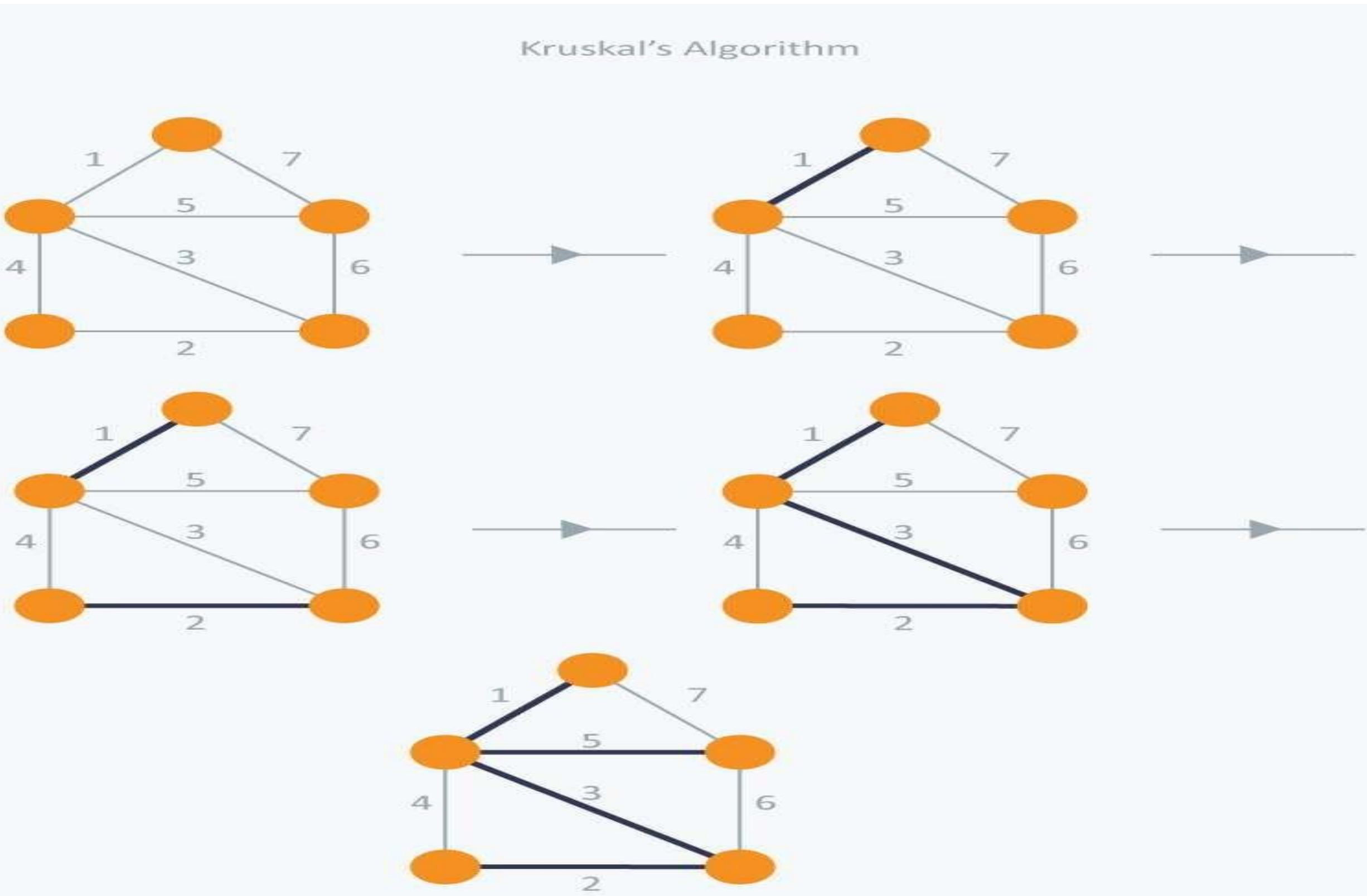
Sort the graph edges with respect to their weights.

Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.

Only add edges which doesn't form a cycle, edges which connect only disconnected components.

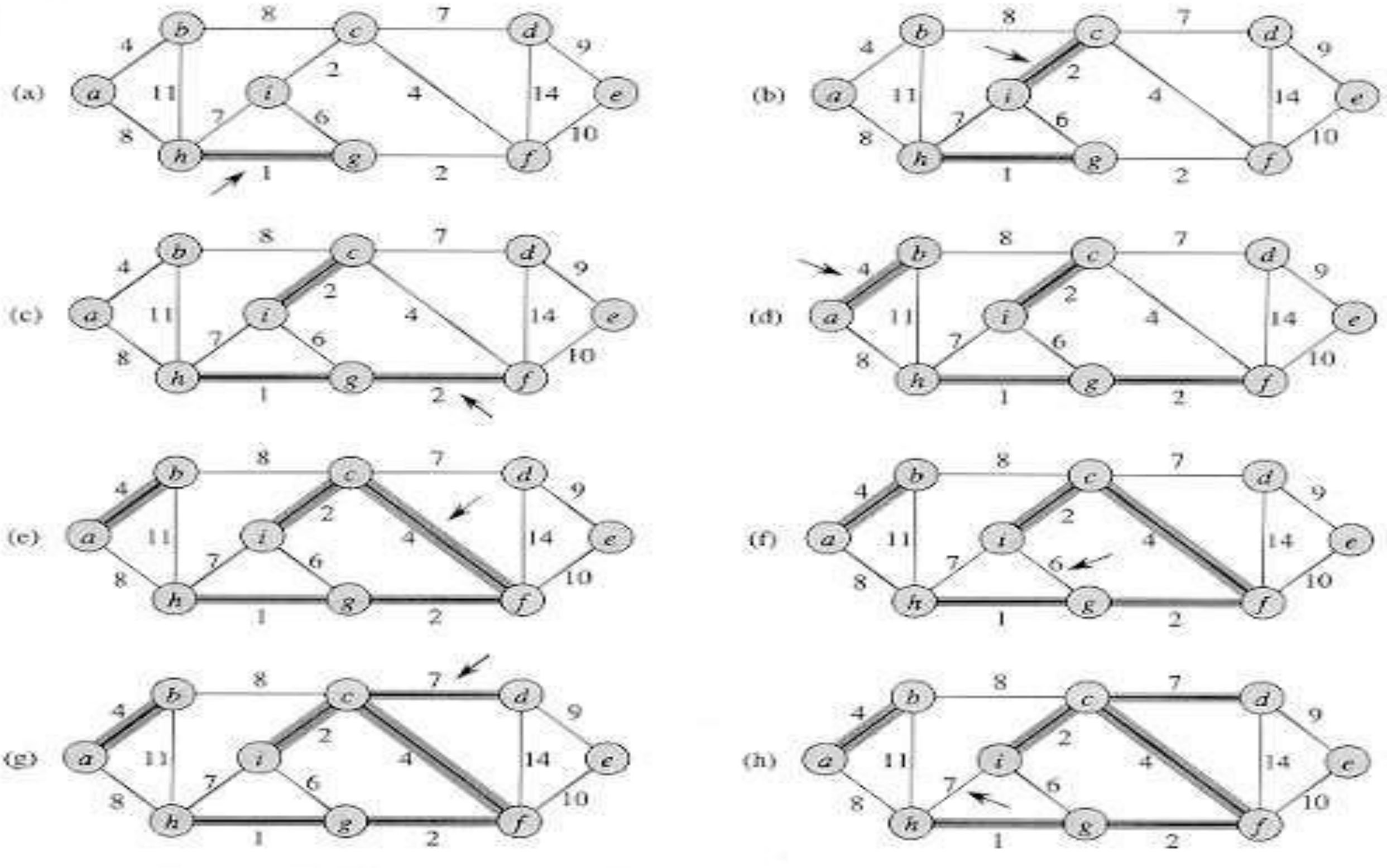
GREEDY STRATEGIES

2. **Kruskal's algorithms : Example**



GREEDY STRATEGIES

2. **Kruskal's algorithms : Example**



GREEDY STRATEGIES

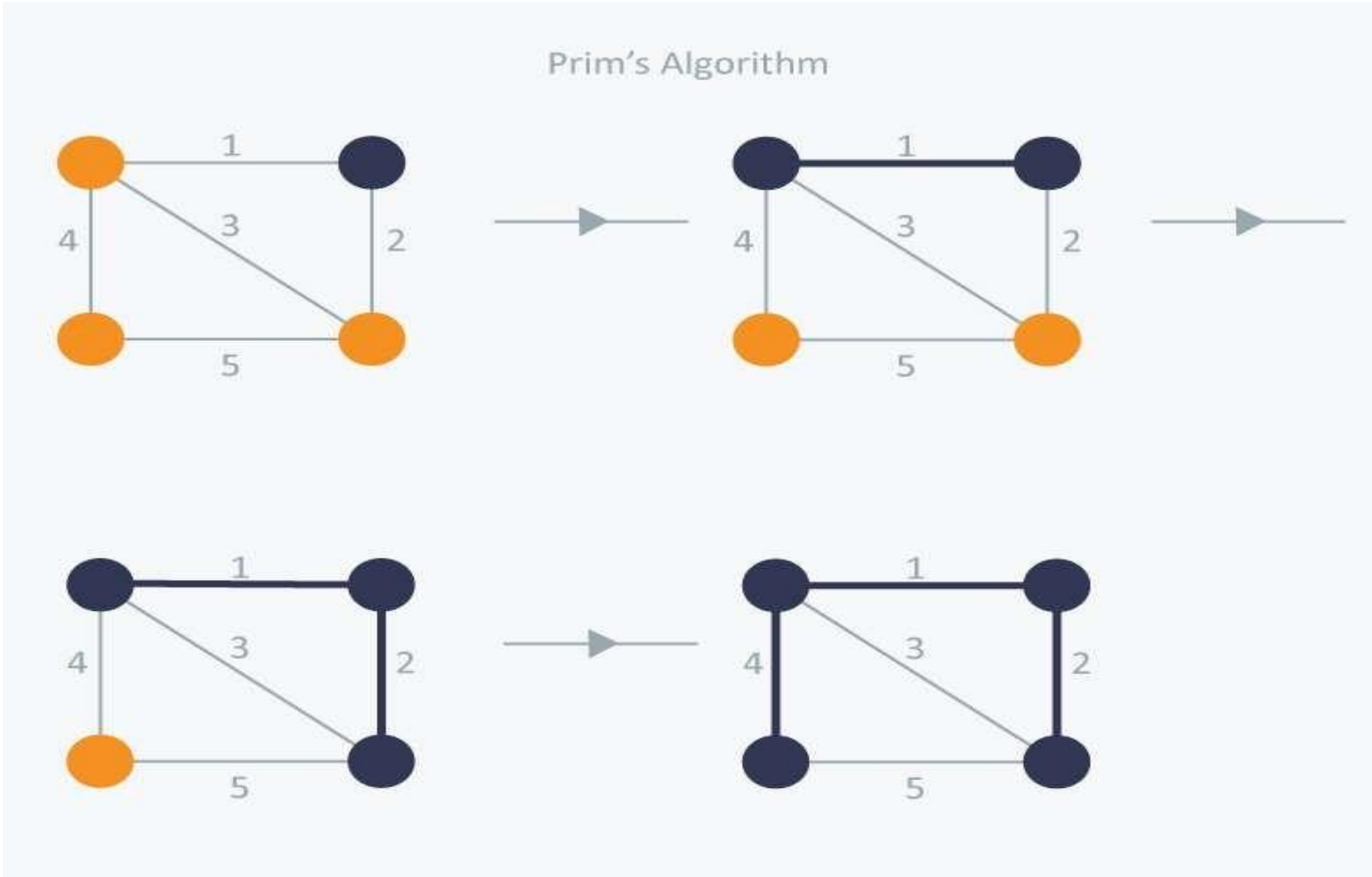
2. **Prims algorithm:** Prim's Algorithm also use Greedy approach to find the minimum spanning tree. In Prim's Algorithm we grow the spanning tree from a starting position. Unlike an **edge** in Kruskal's, we add **vertex** to the growing spanning tree in Prim's.

Algorithm Steps:

1. Initialize the minimum spanning tree with a vertex chosen at random.
2. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree.
3. Keep repeating **step** 2 until we get a minimum spanning tree.

GREEDY STRATEGIES

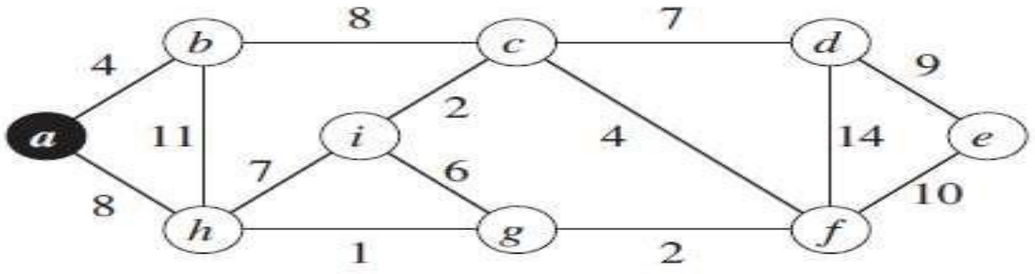
2. Prims algorithm: Example



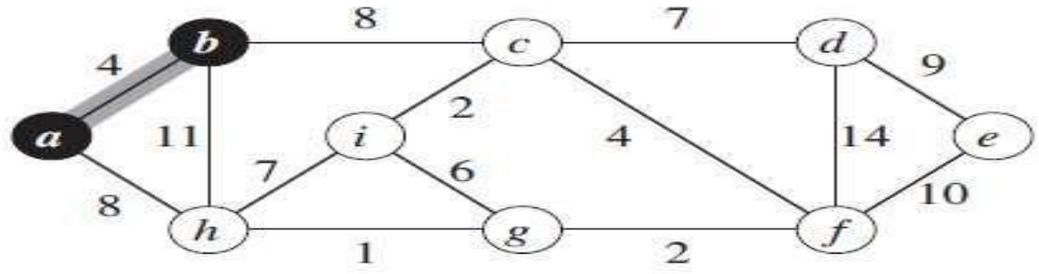
GREEDY STRATEGIES

2. Prims algorithm: Example

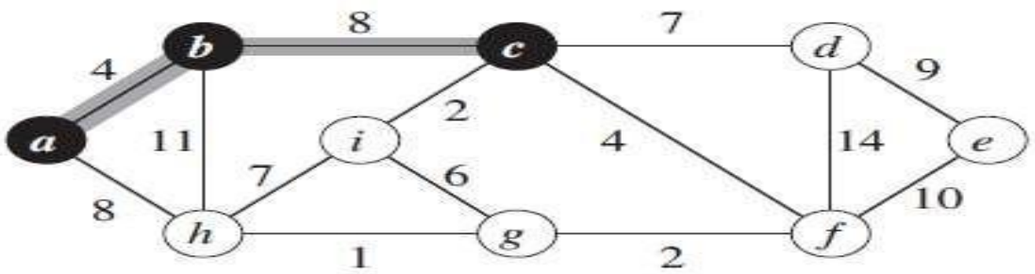
(a)



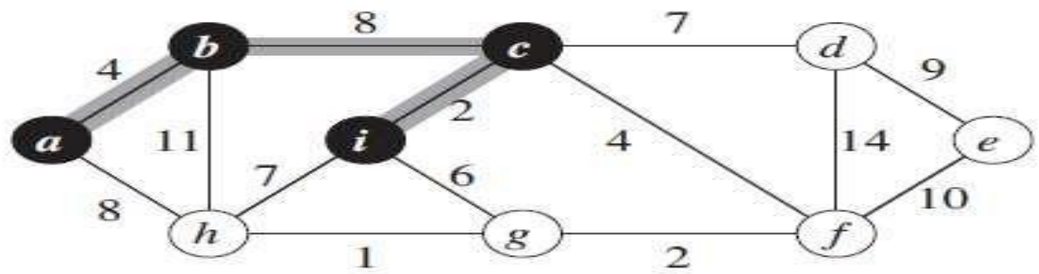
(b)



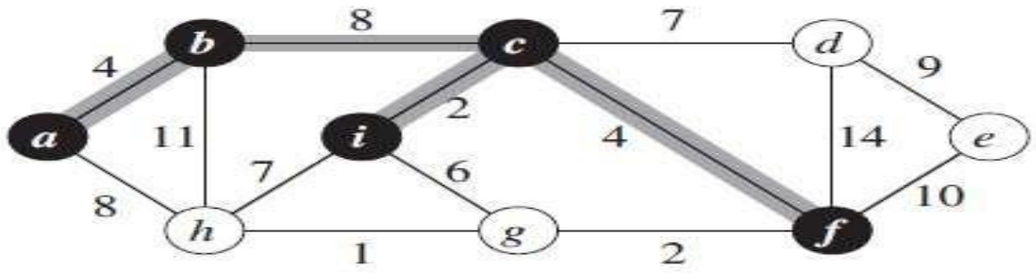
(c)



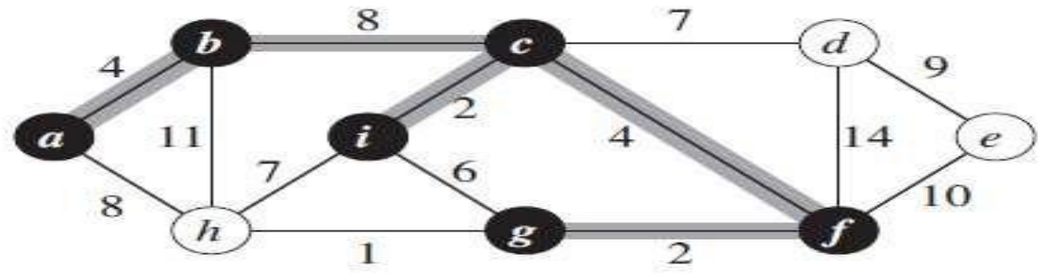
(d)



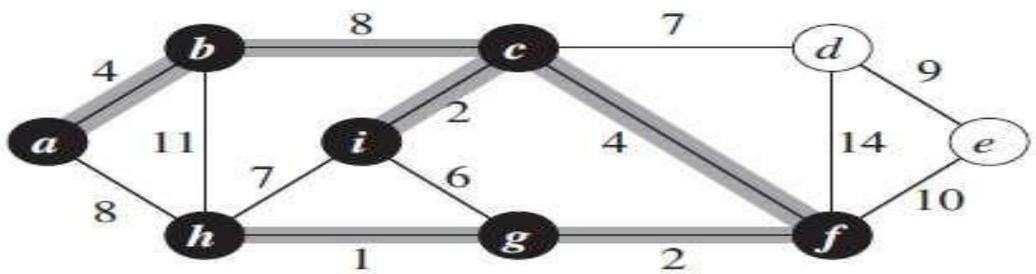
(e)



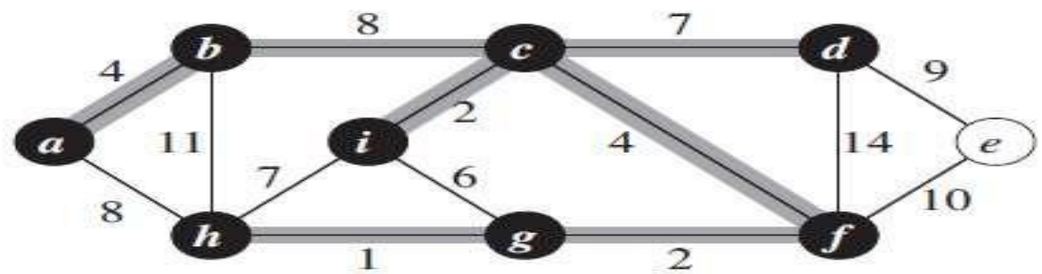
(f)



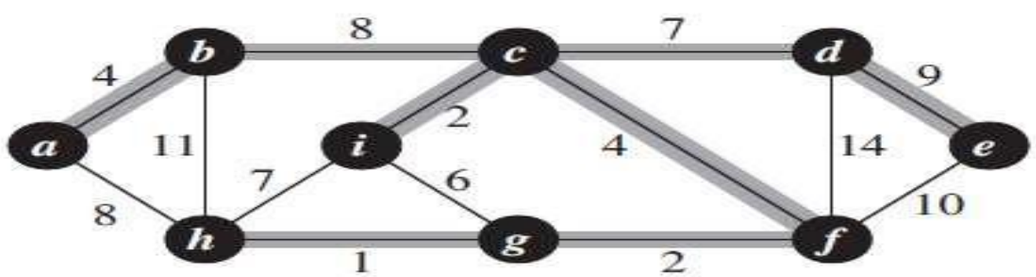
(g)



(h)



(i)



THANK YOU!!!