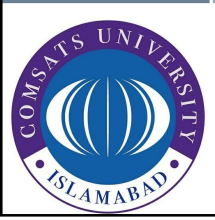
CSC301
Design and Analysis of Algorithm

Dr. Khalid Iqbal
Associate Professor (Tenured)
Department of Computer Science

Lecture 1

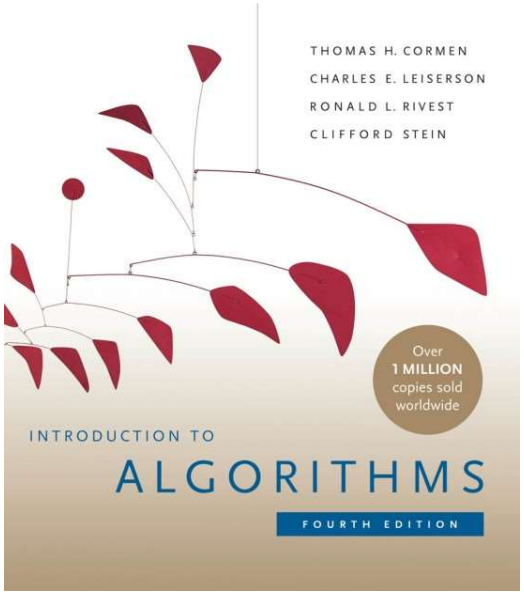
Data Structure: A quick Recap

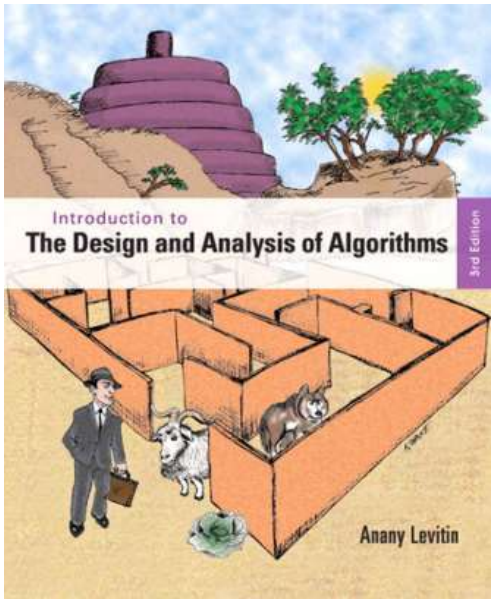
The concept: What are Algorithms, Why Do You Need to Study Algorithms, What Kinds of Problems are Solved by Algorithms, Algorithms as a Technology, How to Represent an algorithm; Features of Algorithms: Input, Output, Precision, Finiteness, Definiteness, Correctness, and Generality



Books

- **Textbook:**
  1. Introduction to the Design and Analysis of Algorithms, Levitin, A., Pearson, 2017.
- **Reference Book:**
  1. Introduction to Algorithms, Cormen, T. H. Leiserson, C.E., Rivest, R.L.& Stein, C., MIT Press, 2019





Course Description File/Assessment

COMSATS University Islamabad
Department of Computer Science
Course Syllabus

Course Information

Course Code: **CSC301**	Course Title: **Design and Analysis of Algorithms**
Credit Hours: **3(3,0)**	Lecture Hours/Week: **3**
Lab Hours/Week: **0**	Pre-Requisites: **CSC211-Data Structures and Algorithms**

Catalogue Description:

This course is designed to provide knowledge of the principles and techniques used in the design and analysis of algorithms. Topics cover: Overview of Algorithm; Proving Correctness of Algorithms; Asymptotic Notations; Solving Recurrence Relations; Sorting & Order Statistics; Brute Force Algorithms & their Analysis; Divide and Conquer; Dynamic Programming; Greedy Algorithms; Graph; and Basic Computability.

Text and Reference Books

Textbook:

1. Introduction to the Design and Analysis of Algorithms, Levitin, A., Pearson, 2017.

Reference Book:

1. Introduction to Algorithms, Cormen, T. H. Leiserson, C.E., Rivest, R.L.& Stein, C., MIT Press, 2019.

Week wise Plan:

Lecture	CDF Unit #	Topics Covered	Reading Material
1.	1	The concept: What are Algorithms, Why Do You Need to Study Algorithms, What Kinds of Problems are Solved by Algorithms, Algorithms as a Technology, How to Represent an Algorithm; Properties of Algorithms: Input, Output, Precision, Finiteness, Definiteness, Correctness, and Generality.	CLRS: Ch1 Levitin: Ch1
2.	1	Fundamentals of Algorithmic Problem Solving: The STAIR Steps for Solving Problems, and Major Factors in Designing an Algorithm.	Levitin: Ch1
3.	2	Pre-condition, Post-condition, Partial Correctness of an Algorithm, Total Correctness of Algorithm, and Illustrative Examples.	Rosen: Ch5
4.	2	Loop Invariants, Correctness of Iterative Algorithm: Initialization, Maintenance, and Termination.	CLRS: Ch3 Thomas: Ch4
5.	2	Correctness of Recursive Algorithm: Quick Review of Mathematical Induction, Proving Correctness of Recursive Algorithm using Induction, and Illustrative Examples.	Thomas: Ch4
6.	3	Time Complexity Measuring Notations: Growth Rate of Functions; Asymptotic Notations: Big O Notation, Sigma Notation, and Theta Notation.	CLRS: Ch3 Levitin: Ch2
7.	3	Major Assumptions in Analyzing Algorithms, Model of Computation (RAM Model), Mathematical Analysis of Non-Recursive Algorithms, and Worst, Best & Average Case Behavior of Algorithms.	Levitin: Ch2
8.	3	Complexity Classes i.e., Constant, Linear, Quadratic; Analysis of Iterative Algorithms, Empirical Measurements of Performance; and Time & Space Tradeoffs in Algorithms.	CLRS: Ch3 Levitin: Ch2
9.	4	Mathematical Analysis of Recursive Algorithms, and Solving Recurrences: Substitution Method.	CLRS: Ch4 Levitin: Ch2
10.	4	Recurrence Tree Method.	CLRS: Ch4 Levitin: Ch2
11.	4	Master Theorem for Solving Recurrences.	CLRS: Ch4 Levitin: Ch2
12.	5	Merge Sort: Overview, Worst, Best & Average Case Analysis; Quick Sort: Overview, and Worst, Best & Average Case analysis.	CLRS: Ch4,Ch7 Levitin: Ch4
13.	5	Randomized Quicksort; Heap Sort: Overview, and Worst, Best & Average Case Analysis.	CLRS: Ch7, Ch6 Levitin: Ch6
14.	5	Sorting in Linear Time: Lower Bounds for Sorting, Counting Sort, and Radix Sort.	CLRS: Ch8 Levitin: Ch7
15.	6	Brute Force Algorithms & their Analysis: Pattern Matching Algorithm & its Time Complexity.	Levitin: Ch3 CLRS: Ch3
16.	6	Brute Force Algorithms & their Analysis: Finding Inversion in an Array, and Closest-Pair & Convex-Hull Problems.	Levitin: Ch3
17.		**Mid Term Exam**	
18.			
19.	7	Dynamic Programming: Comparison of DP, Divide & Conquer, and Elements of Dynamic Programming.	CLRS: Ch15 Levitin, A: Ch8
20.	7	Dynamic Programming (Matrix Chain Multiplication): Problem Analysis, Notations, Designing DP Algorithm for MCM & its Time Complexity, and Applications of MCM.	CLRS: Ch15
21.	7	Dynamic Programming: Optimal Binary Search Trees & its Time Complexity.	CLRS: Ch15 Levitin: Ch8
22.	7	Dynamic Programming (0/1 Knapsack Problem): Problem Analysis, Notations, Designing DP Algorithm for 0/1 Knapsack Problem & its Time Complexity, and Applications of 0/1 Knapsack Problem.	CLRS: Ch15 Levitin: Ch8
23.	7	Dynamic Programming (Longest Common Subsequence): Problem Analysis, Notations, Designing DP Algorithm for LCS & its Time Complexity, and Applications of LCS.	CLRS: Ch15
24.	7	Dynamic Programming (Edit Distance Problem): Edit, Edit Distance Designing DP Algorithm for Edit Distance Problem & its Time Complexity, and Applications Analysis of DP Edit Distance.	CLRS: Ch15
25.	8	Greedy Algorithms: Huffman Encoding & its Time Complexity.	CLRS: 16 Levitin: Ch9
26.	8	Greedy Algorithms: Activity Scheduling, Disjoint Subsets & Union-Find Algorithms.	CLRS: 16
27.	9	Graphs: Terminology, Representation Techniques, Adjacency Matrix, Adjacency List; Various Applications of Graphs, Elementary, Topological Sorting using DFS, and Strongly Connected Component.	CLRS: Ch22
28.	9	Warshall's Algorithm, and Floyd's Algorithm for the All-Pairs Shortest-Paths Problem.	CLRS: Ch24 Levitin: Ch8
29.	9	Maximum Flow Problem & its Working.	CLRS: Ch24 Levitin: Ch10
30.	9	Time Complexity of Maximum Flow Algorithm.	CLRS: Ch24 Levitin: Ch10
31.	10	Introduction to Complexity Classes & Computability.	CLRS: Ch34 Levitin: Ch11
32.	10	NP Complete Problem.	CLRS: Ch34 Levitin: Ch11
		Final Term Exam	

Student Outcomes (SOs)

S.#	Description
1	Apply knowledge of computing fundamentals, knowledge of a computing specialization, and mathematics, science, and domain knowledge appropriate for the computing specialization to the abstraction and conceptualization of computing models from defined problems and requirements
2	Identify, formulate, research literature, and solve *complex* computing problems reaching substantiated conclusions using fundamental principles of mathematics, computing sciences, and relevant domain disciplines
3	Design and evaluate solutions for *complex* computing problems, and design and evaluate systems, components, or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal, and environmental considerations

Course Learning Outcomes (CLO)

Sr.#	Unit #	Course Learning Outcomes	Blooms Taxonomy Learning Level	SO
CLO-1	1-2	Apply inductive proofs techniques to prove the correctness of an algorithm.	*Applying*	1,2
CLO-2	3-5	Analyze the behavior of an algorithm using asymptomatic analysis.	*Analyzing*	2
CLO-3	6-8	Design an algorithm for a computational problem by employing an appropriate algorithmic approach.	*Creating*	2,3
CLO-4	9	Explain the concept of computability with examples.	*Understanding*	1

CLO Assessment Mechanism

Assessment Tools	CLO-1	CLO-2	CLO-3	CLO-4
Quizzes	Quiz 1	Quiz 2	Quiz 3 &4	-
Assignments	Assignment 1	Assignment 2	Assignment 3 &4	-
Mid Term Exam	Mid Term Exam	Mid Term Exam	-	-
Final Term Exam	Final Term Exam			

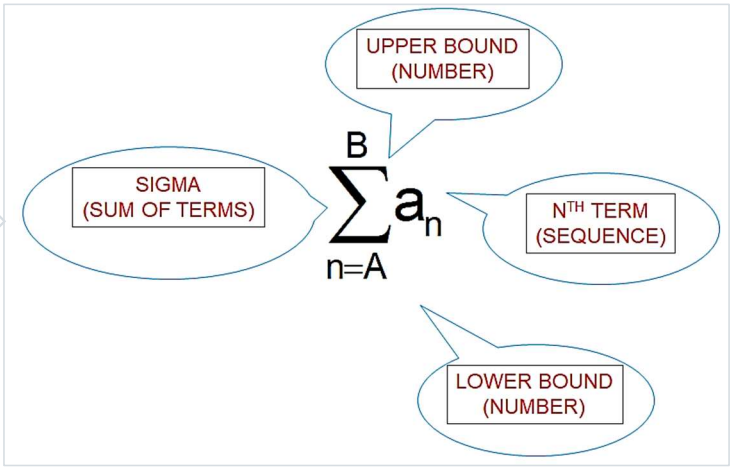
Policy & Procedures

- **Attendance Policy:** Every student must attend 80% of the lectures as well as laboratory in this course. The students falling short of required percentage of attendance of lectures/laboratory work, is not allowed to appear in the terminal examination.

- **Course Assessment:**

	Quizzes	Assignments	Mid Term Exam	Terminal Exam	Final Marks
Theory (T)	15	10	25	50	100

Prior Knowledge
(Discrete Structures)



Data Structure & Algorithm

Idea: Loop syntax, IF Statement, Function & Procedure, Stack, Queue, Tree and Graphs (Know about these terms and functionality)

Recap: Data Structures & Algorithm

- Data refers to the collection of facts and figures based on universal truths
- Ways to Process the Data
  - *Data Structure*
    - *Refer to temporary and manipulation of data*
    - *E.g. Variables and array of a procedural language*
  - *Database*
    - *Refer to Permanent storage and manipulation of data*
    - *E.g. MS Access, Foxpro, Oracle RDBMS, Microsoft SQL Server, MongoDB, etc*
  - *Data structure is way to process and manipulate data through set of operations*

Data Structure Recap?

- *Consider following C language program*

```
Main()
{  int a, b, c;
   a=3; b=5; // Scalar
  // OR cin>>a>>b; // Variable
   c=a+b;
   cout<<"Sum="<<c;
}
```

** Data Stored temporary is our conclusion*

Execute first time:
Output will 8

Execute second and other times.
Again, Output will 8

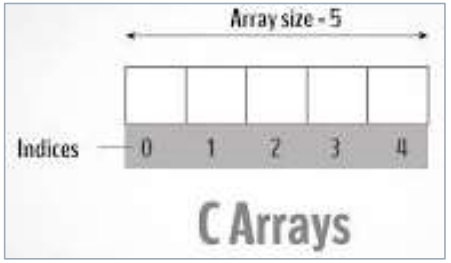
What concluded here

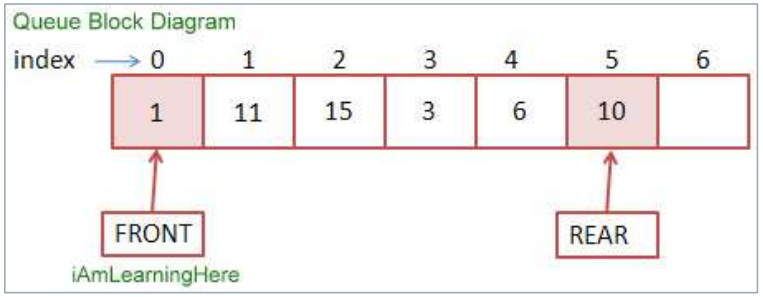
Data Structure Recap?

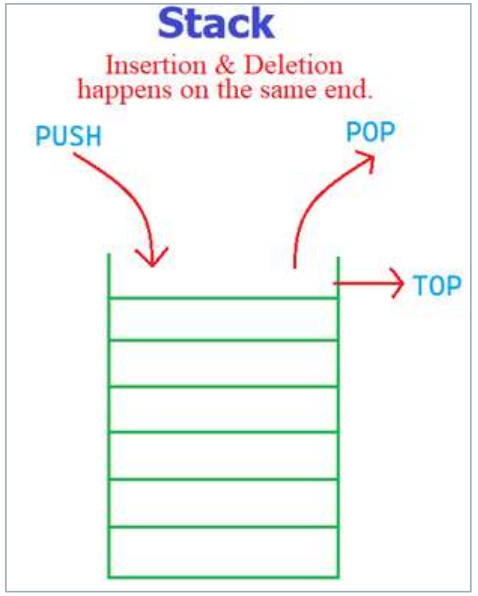
- *Type of Data Structures according to the presentation of data (i.e., How data is presented)*

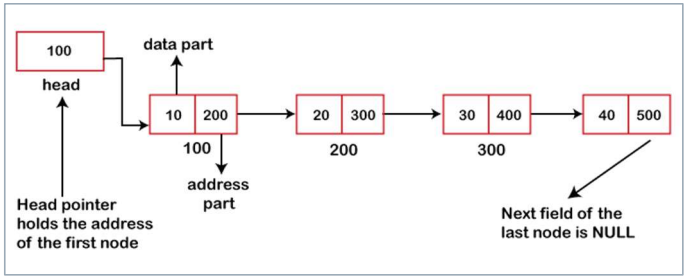
1. *Linear Data Structure : Shows elements in sequence*
    1. *Sequential Data Structures: Combination of homogeneous elements with n consecutive number and successive memory locations*
        1. *Array : Array contains elements of same data type*
        2. *Queue : Queue can contain elements of different data type*
        3. *Stack : Stack has a dynamic and fixed size, and can contain elements of the different data types*
    2. *Pointer Data Structure (Linked List) : Connecting elements with addresses*
2. *Non-Linear Data Structure: data elements are not arranged sequentially or linearly*
    1. *Tree : A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.*
    2. *Graphs : A non-linear data structures made up of a finite number of nodes or vertices and the edges that connect them*

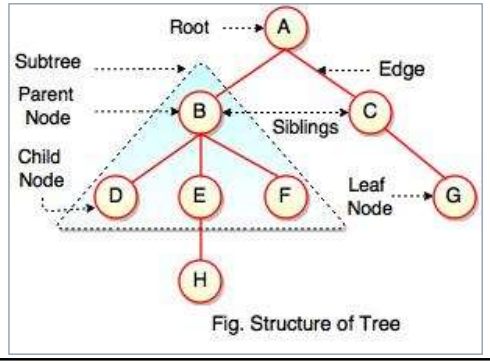
Data Structure Recap?

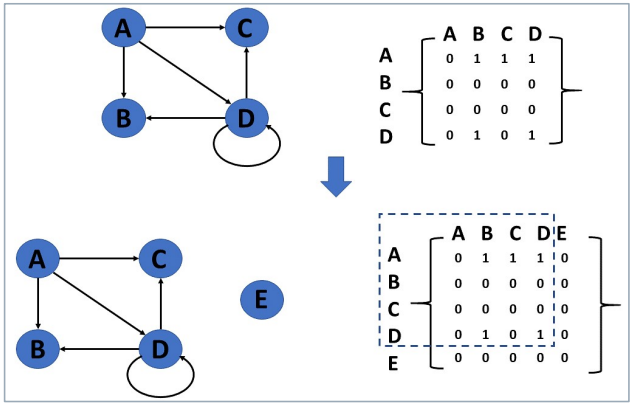






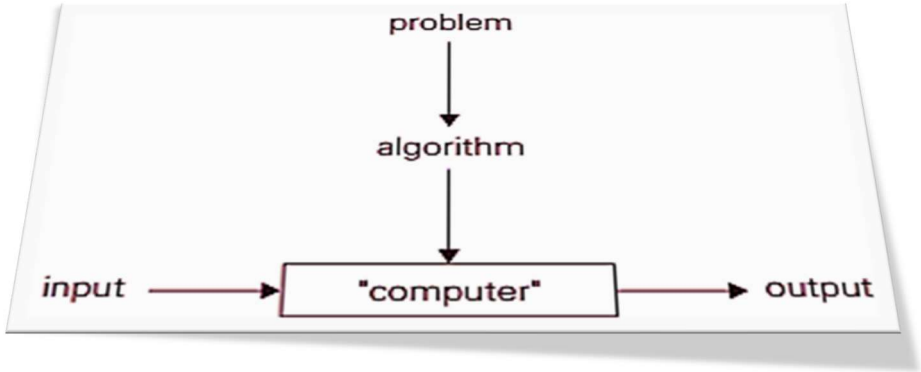






What is an Algorithm?

- An algorithm is a sequence of instructions that one must perform in order to solve a well-formulated problem.
- An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.
- An algorithm is thus a sequence of computational steps that transform the input into the output.
- We will specify problems in terms of their inputs and their outputs, and the algorithm will be the method of translating the inputs into the outputs.
- A well-formulated problem is unambiguous and precise, leaving no room for misinterpretation.



What is an algorithm?

- An algorithm is a step-by-step procedure to solve a problem
  - *Plate form independent*
  - *Can be implement in any procedural language*
- *Simplest form (for experts) is **Pseudo code***
- *Pseudo code can be written in any language syntax, but here in this course we will used C/C++ syntax*
- *Two important factors*
  - `Space Tradeoff` *(Refer to less use of memory i.e., RAM)*
  - `Time Tradeoff` *(Refer to less time of Micro processor for execution*

What kinds of Problems are solved by Algorithms?

- The Human Genome Project (determining the sequences of the 3 billion chemical base pairs that make up human DNA)
- Finding good routes on which the data will travel (Internet)
- Privacy of personal information such as credit card numbers, passwords, and bank statements.(E-Commerce)
- To allocate scarce resources in the most beneficial way (Manufacturing enterprises e.g. An airline may wish to assign crews to flights in the least expensive way possible)

Features of algorithm

- ***Complete*** *: An algorithm is complete* ***if it guarantees to return a correct answer*** *for any arbitrary input (or, if no answer exists, it guarantees to return failure).*
- ***Finite*** *: An algorithm is a finite sequence of instructions for performing a task. By finite we mean that there is* ***an end to the sequence of instructions****.*
  - ***0,1 or more inputs*** *: 1 cup of milk, 2 tablespoons of sugar, 1 cup of flour, 3 large eggs, 1 pinch of salt, and oil for the pan*
  - ***At least one Output*** *: a stack of pancakes on a plate*
- *Correct: An algorithm is correct only if it* ***produces correct result for all input instances***
- *Clarity: It refers to how well it is expressed and described*

An Algorithm (Example)

Algorithm SUM (No1, No2, Res)

{ This algorithm is used to read two numbers and print its sum } – *Introductory comments*

Step-1 [Read numbers No1 and No2]

Read(No1,No2)

Step-2 [Calculate the sum]

Res=No1+No2

Step-3 [Display the Result]

Write(Res)

Step-4 {Finish]

Exit

C Language Program:

```
void main()
{
  int no1, no2, res;
  scanf("%d%d", &a, &b);
  res = a+b;
  printf("Sum=%d", res);
 }
```

Algorithms Notations

- *Algorithm Name*
  - *Should be in capital form*
  - *Meaningful*
- *Parameters*
  - *Should be enclosed in parenthesis ( )*
  - *Variable name comprises on more than one characters*
  - *Scripting or looping variable are of single character*
- *Introductory Comment*
  - *Description and purpose of an algorithm*
- *Steps*
  - *Finite steps*

Algorithms Notations (Cont !!!)

- *Comments*
  - *Each step start with a comment*
  - *Enclose in [ ]*
  - *Define the purpose of step*
- *Input Statements*
  - *Read*
  - *Scanf (if using C/C++)*
- *Output statement*
  - *Write*
  - *Printf (if using C/C++)*

Algorithms Notations (Cont !!!)

- *Selection statement*
  - *If –Then –End If*
  - *If – then ---Else --- End If*
  - *Nested If*
  - *Example*

```
If ( a>b ) then
  write ( a+"Is Large")
Else
  write ( b+"Is Large")
End if
```

Algorithms Notations (Cont !!!)

- *Loops (For, While, Until)*
  - *Example-1*

    Repeat Step 2 For i=1, N, 1
  - *Example-2*

    Repeat Step 2 to Step 4 For i=1, N, 1
  - *Example-3*

    Repeat Step 2 while i<=10
  - *Example-4*

    Repeat Step 2 Until i>10

Note:- Purpose of all examples is same i.e. to perform loop 10 times

Algorithms Notations (Cont !!!)

- *Finish*
  - *Exit (Used in main algorithm)*
  - *Return (Used in sub algorithm)*

Example-1.

- *Write an algorithms or Pseudo code to read two number and display the largest number.*

Algorithm Example-1.

Algorithm LARGE(No1, No2, lar)

{ This algorithm is used to read two numbers and print the largest}

Step-1 [Read numbers No1 and No2]

Read(No1,No2)

Step-2 [Find the largest]

if (No1 > No2) then

lar = No1

else

lar = No2

end if

Algorithm Example-1 (Cont !!!)

Step-3 [Display the Result]

Write(lar)

Step-4 {Finish]

Exit

Algorithm Example-1 (Cont !!!)

Write the algorithms convention which are use in Example-1.

- *Algorithm Name (Large)*
- *Parameters (No1, No2 and lar)*
- *Input and Output Statement(Read, Write)*
- *Selection Statement (if-else)*
- *Comments (enclosed in [ ] before start of each step)*
- *Introductory comments (enclosed in { })*
- *Finish (exit)*

Pseudo code Example-1 (Cont !!!)

```
Large (a, b, lar)
{
// Find the largest number
  if (a>b)
    lar=a;
else
  lar=b;
}
```

Example-2.

- *Write an algorithms or Pseudo code to read an array of N integer values, calculate its sum and then print the sum?*

Algorithm Example-2.

Algorithm SUM(Ary[ ], I, total, N)

*{ This algorithm is used to read an array **Ary** of size **N** and display its **sum** }*

Step-1 [Perform a loop to read the values of array]

Repeat step 2 for i=1,N,1

Step-2 [Read value]

Read(Ary[i])

Step-3 [Initialize variable I and total]

a. I =1

b. total = 0

*Step-4 [Perform a loop to traverse the all elements of array **ary** and add]*

Repeat step 5 while i<=10

Algorithm Example-2 (Cont !!!)

Step-5 [Add the values]

total = total + ary [I]

Step-6 {Display the sum of values]

Write (total)

Step-7 {Finish]

Exit

Algorithm Example-2 (Cont !!!)

Write the algorithms convention which are use in Example-2.

- *Algorithm Name (SUM)*
- *Parameters (ary[], I, N, total)*
- *Input and Output Statement (Read, Write)*
- *Loop Statement (for, while)*
- *Assignment*
- *Comments (enclosed in [ ] before start of each step)*
- *Introductory comments (enclosed in { })*
- *Finish (exit)*

Pseudo code Example-2(Cont !!!)

```
Large (Ary, N, I, Total)
{
for(i=0;i<=N-1;i++)
  scanf("%d", Ary[i]);
  i=0; total=0;
    while(i<=N-1)
        total=total+ Ary[i];
    printf("%d", total);
}
```

Example-3.

- *Write an algorithms or Pseudo code to find the factorial of number N using sub algorithm approach.*

Algorithm Example-3.

Algorithm FACTORIAL(No, Res)

{ This algorithm is used to read a number ***No*** *and print its* ***factorial*** *}*

Step-1 [Read the number No]

Read (No)

Step-2 [Call Sub algorithm FACT]

Call Res = FACT(No)

Step-3 [Display factorial]

Write (Res)

Step-4 [Finish]

Exit

SubAlgorithm FACT(NewNo, i, f)

{ This Sub algorithm is used to receive an element from main algorithm FACTORIAL and return the result }

Step 1. [Initialize variable]

f =1

Step2. [Perform loop to calculate factorial]

Repeat step 3 for i=1, NewNo, 1

Step 3. [Calculate f]

*f = f * i*

Step 4 [Finish]

return (f)

Algorithm Example-3 (Cont !!!)

Write the algorithms convention which are use in Example-3.

- *Algorithm **Name** (FACTORIAL)*
- *Sub Algorithm **Name** (FACT)*
- ***Parameters** of FACTORIAL (No, Res)*
- ***Parameters** of FACT (NewNo, F, I)*
- ***Input** and **Output** Statement (Read, Write)*
- ***Loop** Statement (for, while)*
- ***Assignment***
- ***Comments** (enclosed in [ ] before start of each step)*
- ***Introductory comments** (enclosed in { })*
- ***Finish** (exit, return)*

Pseudo code Example-3(Cont !!!)

```
Facorial(No, Res)
{
Res= Fact (No)
Printf("%d", Res)
}
```

```
Fact(NewNo)
{
for(i=1;i<=N;i++)
F = F *  I
Return(f)
}
```

Summary

- *An algorithm is a step-by-step process to solve a problem.*
- *An algorithm is platform Independent, and you can make a computer program in any language.*
- *No of Inputs, outputs, completeness, accuracy , correctness and finite are the main features of algorithms*
- *Algorithms notation are used to design an algorithm*
- *Pseudo code are used by experts*

Homework

- *Write an algorithm which read three numbers and print the smallest number. Also write a C++ and Python language program?*
- *Write an algorithm which read an array of 10 integers and count the even numbers. Also write a C++ and Python language program?*
- *Write an algorithm which read two values and find its product using a sub algorithm. Also write a C++ and Python language program?*

Thank You!!!

Have a good day