What is Data?

1

Data are the raw facts, unorganized facts that need to be processed. Data can be something simple and seemingly random and useless until it is organized.

What is Information?

When data is processed, organized, structured or presented in a given context so as to make it useful, it is called information.

Difference between Data & Information

BASIS FOR COMPARISON	DATA	INFORMATION
Meaning	Data means raw facts gathered about someone or something, which is bare and random.	Facts, concerning a particular event or subject, which are refined by processing is called information.
What is it?	It is just text and numbers.	It is refined data.
Based on	Records and Observations	Analysis
Form	Unorganized	Organized
Useful	May or may not be useful.	Always
Specific	No	Yes
Dependency	Does not depend on information.	Without data, information cannot be processed.

Algorithm, Program & Data Structure?

Algorithm Outline, the essence of a computational procedure, step by step instructions

Program An Implementation of Algorithm in some programming language

Data Structure Organization of data needed to solve the problem

Algorithm V/s Pseudocode

Algorithm	Pseudocode
Systematic logical approach which is a well-defined, step-by-step procedure that allows a computer to solve a problem.	It is one of the methods which can be used to represent an algorithm for a program.
Algorithms can be expressed using natural language, flowcharts, etc.	Pseudocode allows you to include several control structures such as While, If-then-else, Repeat-until, for and case, which is present in many high-level languages.

Data Structure & Algorithm

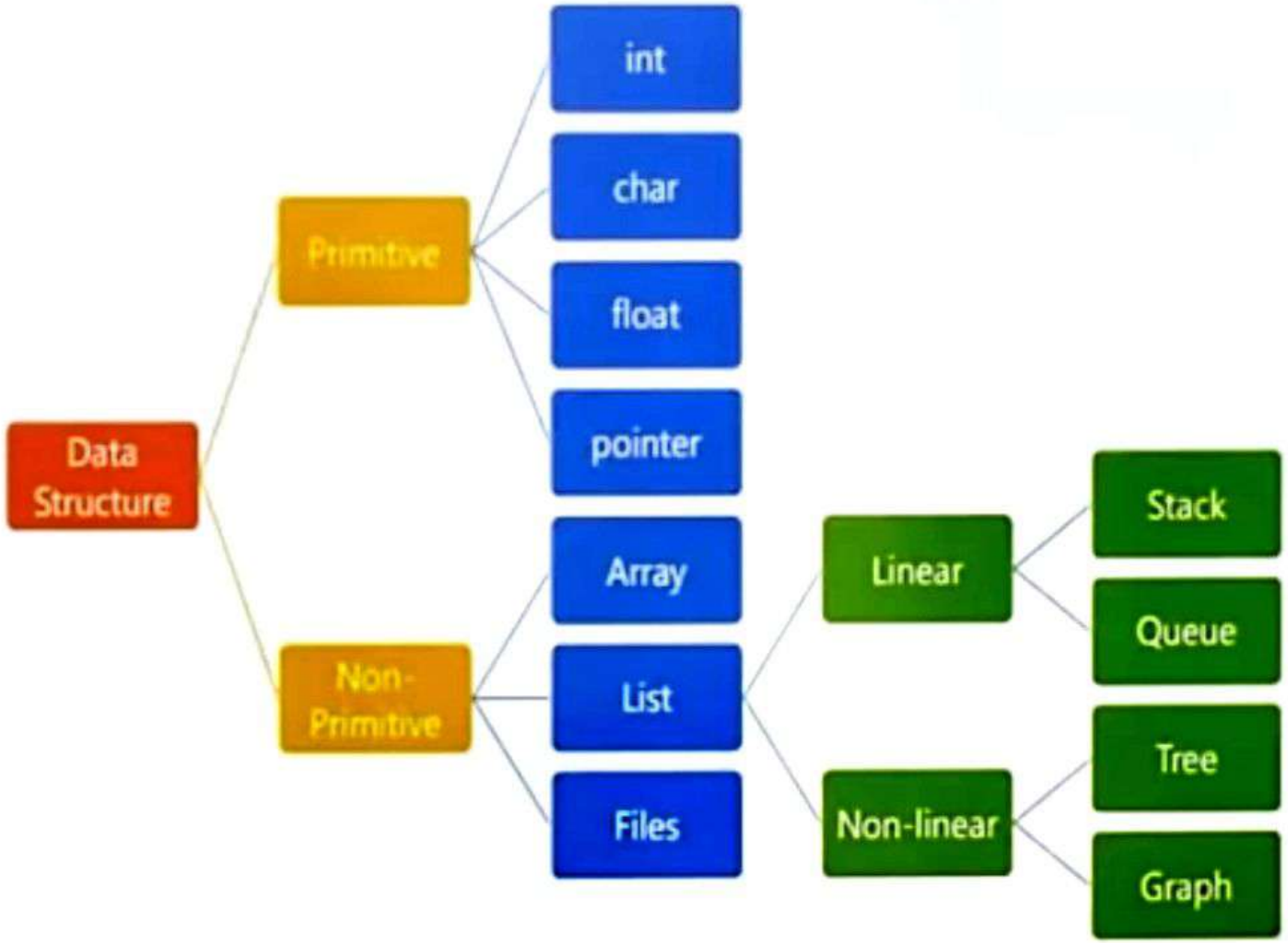
Data Structures

Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way.

Types Of Data Structures

- Primitive data structures
- Non-primitive data structure

Classification of Data Structures



Primitive and Non-primitive DS

Primitive Data Structures: Primitive Data Structures are the basic data structures that directly operate upon the machine instructions.

Non-primitive Data Structures

Non-primitive data structures are more complicated data structures and are derived from primitive data structures.

They emphasize on grouping same or different data items with relationship between each data item.

linear and non-linear data structure

Linear DS:

- every item is related to its previous and next time.
- data is arranged in linear sequence.
- data items can be traversed in a single run.
- implementation is easy

Non-linear DS:

- every item is attached with many other items.
- data is not arranged in sequence.
- data cannot be traversed in a single run.
- implementation is difficult.

Static and Dynamic DS

Static	Static data structures are those whose sizes and structures associated memory locations are fixed, at compile time. Example: **Array**
Dynamic	Dynamic structures are those which expands or shrinks depending upon the program need and its execution. Also, their associated memory locations changes. Example: **Linked List created using pointers**

Homogeneous and Non-Homogeneous DS

Homogeneous	In homogeneous data structures, all the elements are of same type. Example: **Array**
Non-Homogeneous	In Non-Homogeneous data structure, the elements may or may not be of the same type. Example: **Structures**

Data Structure & Algorithm

Asymptotic Notation

Introduction

Algorithm: Outline, the essence of a computational procedure, step by step instructions

Program: An Implementation of Algorithm in some programming language

Data Structure: Organization of data needed to solve the problem

What is a Good Algorithm?

Efficiency

- Running Time
- Space used

Measuring the running time

Experimental Study

- Write a program that implements an algorithm
- Run the program with data sets of varying size
- Use the method like system.current.timemillis() to get an accurate measure of the actual running time.

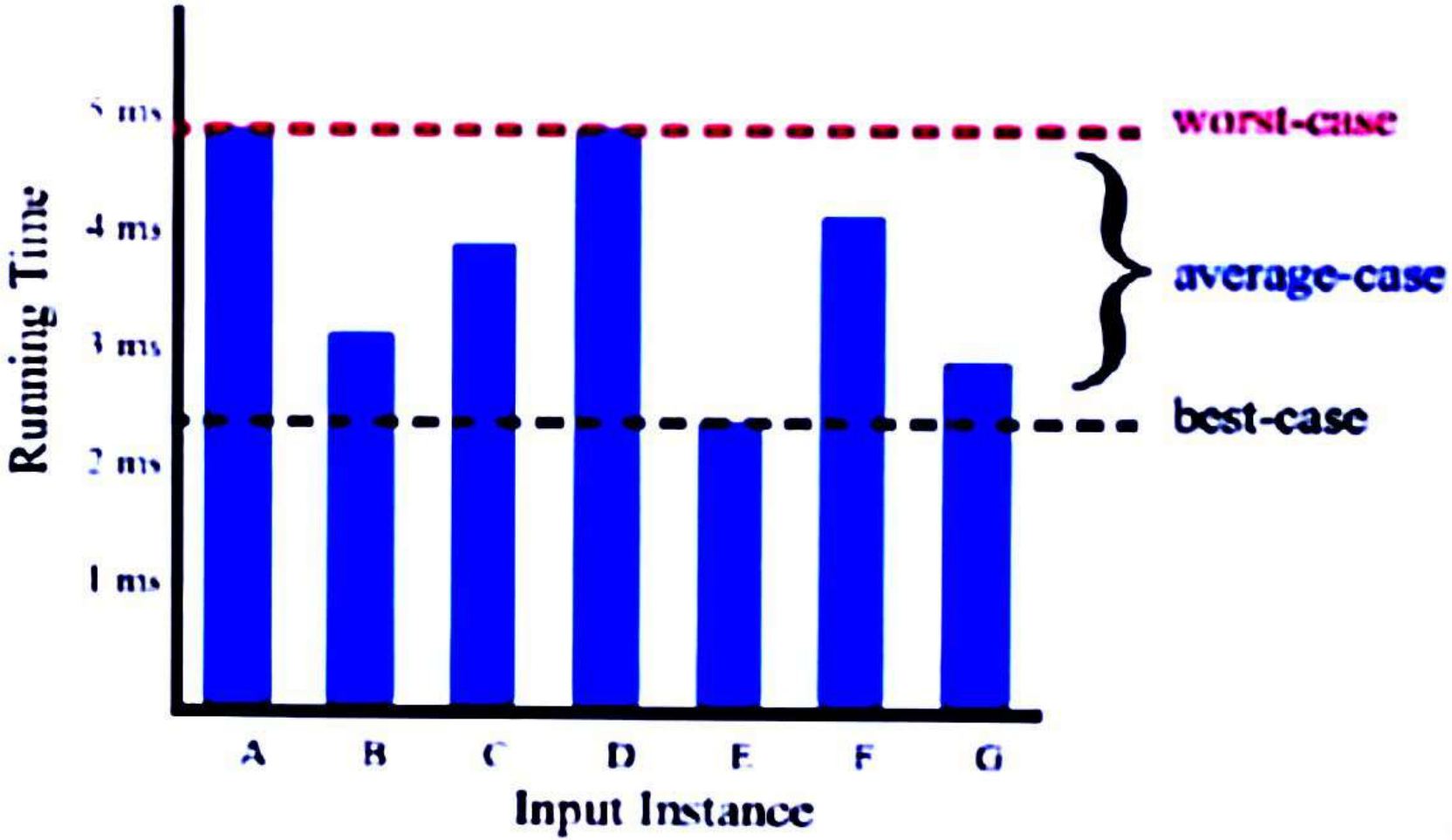
Limitations of Experimental Study

- It is necessary to implement and test the algorithm in order to determine its running time.
- Experiments can be done only on a limited sets of inputs .
- In order to compare algorithms, the same set of hardware and software should be used.

Beyond Experimental Study

- We develop a general methodology for analyzing running time of algorithms.
- Uses high level description of algorithm instead of testing one of its implementation.
- Takes into account all possible inputs.
- Allows one to evaluate the efficiency of any algorithm in a way that is independent of hardware and software environment.

Best/Worst/Average Case



Asymptotic Analysis

- To Simply the analysis of running time by getting the rid of details which may be affected by specific implementation hardware.
- Capturing the essence: how the running time of an algorithm increases with the size of the input in the limit.

Two Basic Rules:-

- Drop all lower order terms
- Drop all constants

Asymptotic Notations

Asymptotic notation of an algorithm is a mathematical representation of its complexity.

There are three types of Asymptotic Notations...

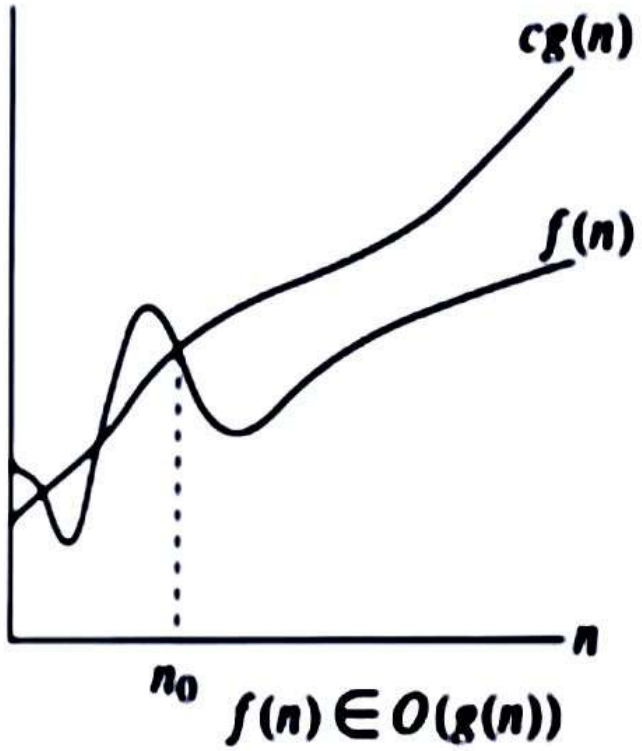
- Big - Oh (O)
- Big - Omega (Ω)
- Big - Theta (Θ)

Big - Oh Notation (O)

- Big - Oh notation is used to define the upper bound of an algorithm in terms of Time Complexity. It provides us with an **asymptotic upper bound.**
- That means Big - Oh notation always indicates the maximum time required by an algorithm for all input values. That means Big - Oh notation describes the worst case of an algorithm time complexity.

Big - Oh Notation (O)

- f(n) is your algorithm runtime, and g(n) is an arbitrary time complexity you are trying to relate to your algorithm. f(n) is O(g(n)), if for some real constants c (c > 0) and n0, f(n) <= c g(n) for every input size n (n > n0).

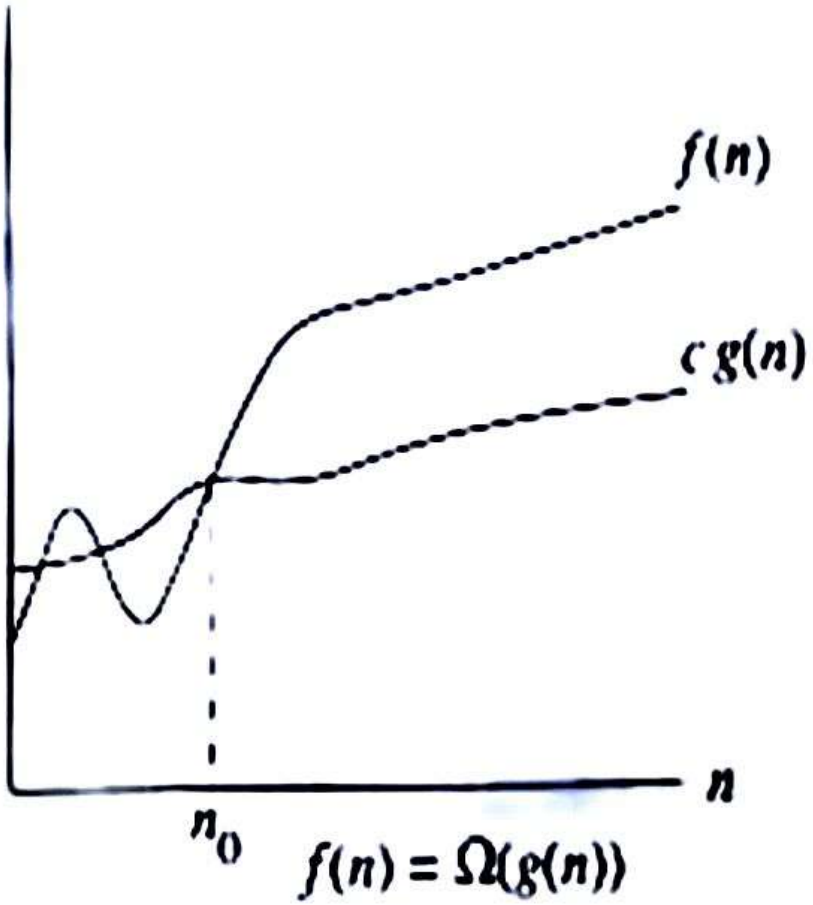


Big - Omege Notation (Ω)

- Big - Omega notation is used to define the lower bound of an algorithm in terms of Time Complexity.
- That means Big-Omega notation always indicates the minimum time required by an algorithm for all input values. That means Big-Omega notation describes the best case of an algorithm time complexity. It provides us with an **asymptotic lower bound**.

Big - Omege Notation (Ω)

- f(n) is Ω(g(n)), if for some real constants c (c > 0) and n0 (n0 > 0), f(n) is >= c g(n) for every input size n (n > n0).

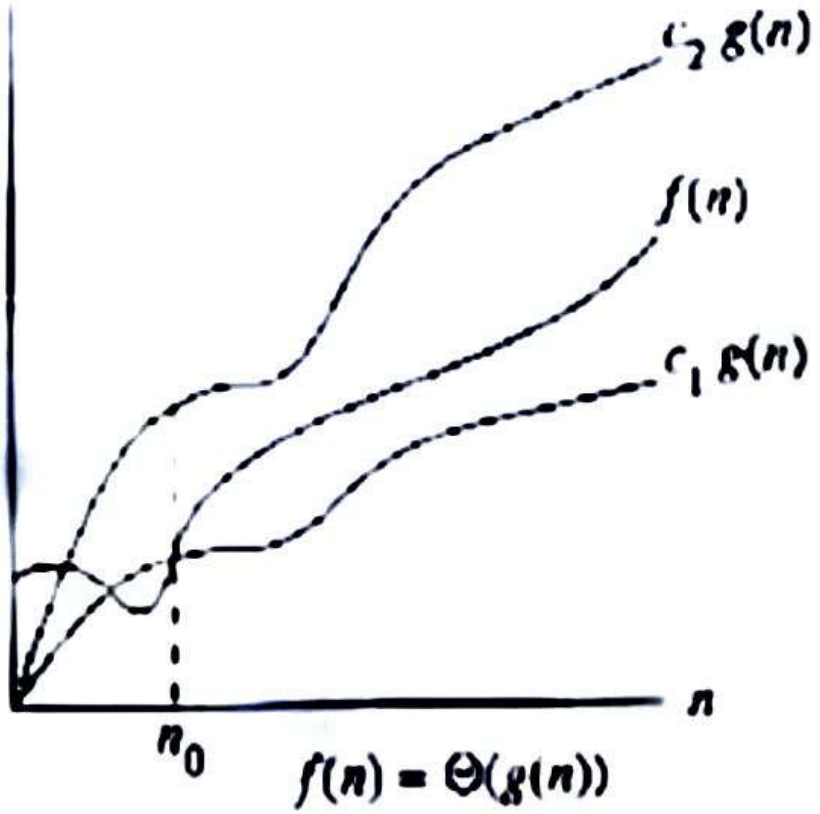


Big - Theta Notation (Θ)

- Big - Theta notation is used to define the average bound of an algorithm in terms of Time Complexity and denote the ***asymptotically tight bound***
- That means Big - Theta notation always indicates the average time required by an algorithm for all input values. That means Big - Theta notation describes the average case of an algorithm time complexity.

Big - Theta Notation (Θ)

- function f(n) as time complexity of an algorithm and g(n) is the most significant term. If C1 g(n) <= f(n) <= C2 g(n) for all n >= n0, C1 > 0, C2 > 0 and n0 >= 1. Then we can represent f(n) as Θ(g(n)).



Common Asymptotic Notations

NOTATION	NAME
$O(1)$	constant
$O(\log n)$	logarithmic
$O(n)$	linear
$O(n^2)$	quadratic
$O(n^c)$	polynomial or algebraic
$O(c^n)$; where $c>1$	exponential

Summary

Analogy with real numbers

$$f(n) = O(g(n)) \quad \cong \quad f \leq g$$

$$f(n) = \Omega(g(n)) \quad \cong \quad f \geq g$$

$$f(n) = \Theta(g(n)) \quad \cong \quad f = g$$

$$f(n) = o(g(n)) \quad \cong \quad f < g$$

$$f(n) = \omega(g(n)) \quad \cong \quad f > g$$