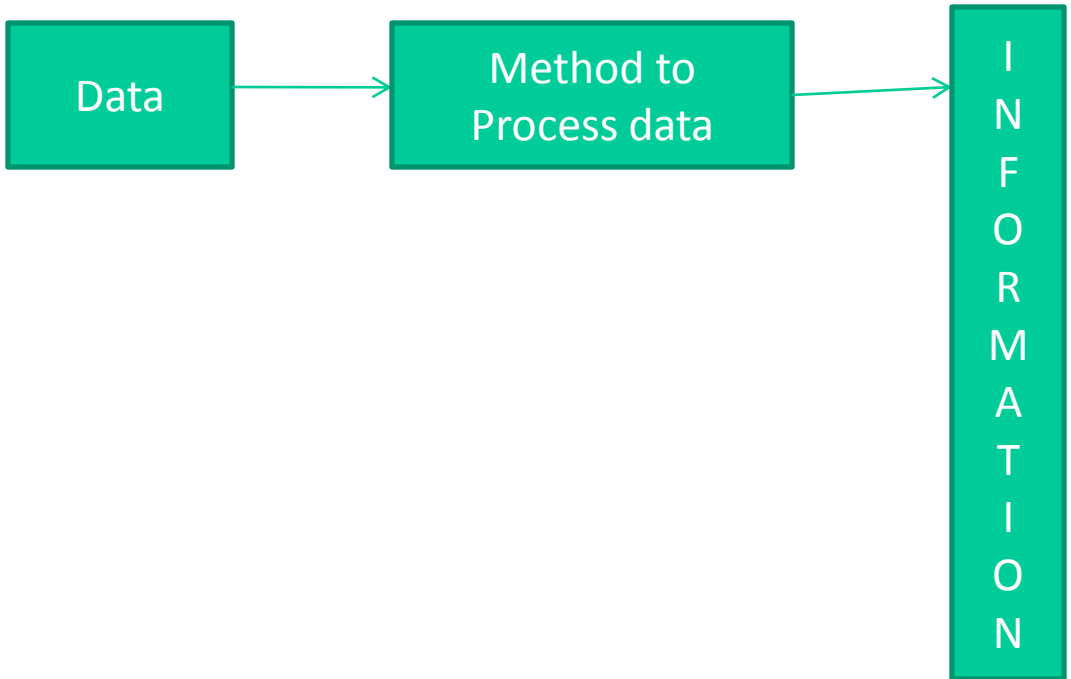
Introduction to Data Structures

Introduction

- Data means value or set of values . Data items refers to a single unit of values.
- Data is represented in the from of text/numbers/figures/ tables/graphs/pictures etc. A computer language is incomplete without data.
- Information is derived from data, such as:

  per capita income, Average Population of the states , price index ,etc.

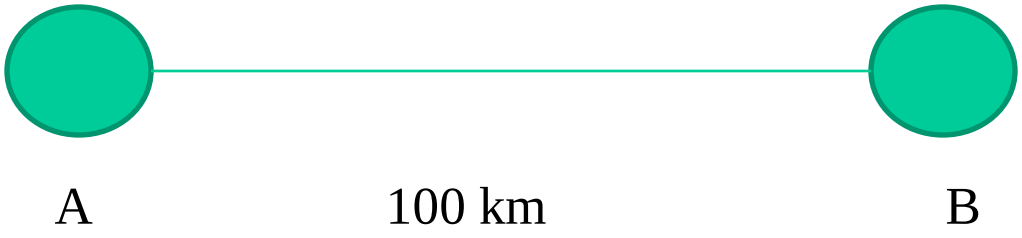


Introduction

- In a month we everyday get the newspaper then at the end we can calculate the bill from the data of the cost of newspaper we daily receive.
- Data about some persons in a group:

Information Details	Data
Name	Satish
Gender	Male
Age	32
Phone	987680987

- Distance between the two stations:



Introduction

- From the score of an individual cricket player we can draw following information:
  - Highest score
  - Total Runs Scored
  - Average
  - Strike rate
  - No. of Sixes
  - No. of fours
- This information helps the planners/analysts/to make decisions.
- Info. in Computer: A computer is an electronic device that manipulates the information presented in the form of data.
- Data: Data is nothing but a collection of numbers, alphabets and symbols combined to represent information.
- Entity: An entity possesses some attributes and some values can be allocated to it. E.g. student entity in a college.

Overview of Data Structures

- Data Structures are a method of representing of logical relationships between individual data elements related to the solution of a given problem.
- A data structure is a structured set of variables associated with one another in different ways, co-operatively defining the components in the system and capable of being operated upon in the program.
- Data structures are the basis of the programming tools. The choice of data structures should provide the following:

1. The data structure should satisfactorily represent the relationship between the data elements.
2. The data structure should be easy so that the programmer can easily process the data.

Classification of Data Structures

- **Linear:** The values are arranged in a linear fashion. E.g. Arrays, Linked Lists, Stacks, Queues etc.
- **Non-Linear:** The values are not arranged in an order. E.g. Tree, Graph, Table etc.
- **Homogeneous:** Here all the values stored are of same type e.g. arrays
- **Non- Homogeneous:** Here all the values stored are of different type e.g. structures and classes.
- **Dynamic:**A *dynamic* data structure is one that can grow or shrink as needed to contain the data you want stored. That is, you can allocate new storage when it's needed and discard that storage when you're done with it. E.g. *pointers,* or *references*
- **Static:**They're essentially fixed-size and often use much space E.g. Array

Basic Notations

Algorithmic Notations:

- Finite step by step list of well-defined instructions for solving a particular problem.
- Formal presentation of the Algorithm consists of two parts:
    1. A paragraph that tells the purpose of an algorithm. Identifies the variables which occur in the algorithm and lists the input data.
    2. The list of the steps that is to be executed.
- Some conventions used in Algorithms:
    - Identifying Number (Algorithm 2:)
    - Steps, Control and Exit (All statements are numbered, Go to and Exit)
    - Comments (The comments are to be given to understand the meaning)
    - Variable Names (capital letters are used for variable names)
    - Assignment Statement (Max:=DATA[1])
    - Input and output (Read: variable names, Write: Messages and/or variable names)
    - Procedure

Basic Notations

Control Structures:

1. Sequential Logic or Sequential Flow
2. Selection Logic or Conditional Flow
   - If (condition) Endif
   - If (condition) Else Endif
   - Multiple If Elseif Else Endif
3. Iteration Logic (Repetitive Flow)

   Repeat for K=R to S by T:

   [Module]

   [End of Loop]

Types of Data Structures

Arrays:

1. Linear data structure
2. All elements stored at contiguous memory locations
3. Every element is identified by an index (Logical Address)
4. At the same time every element is identified by a physical address (Physical Address)
5. We can have 2D and 3D arrays also but stored in memory in a linear way.

Types of Data Structures

Stack:

- Only the top item can be accessed
- Can only extract one item at a time

- A stack is a data structure with the property that only the top element of the stack is accessible
- The stack's storage policy is Last-In, First-Out
- Only the top element of a stack is visible, therefore the number of operations performed by a stack are few
- Needs the ability to:
  - Inspect the top element
  - Retrieve the top element
  - Push a new element on the stack
  - Test for an empty stack

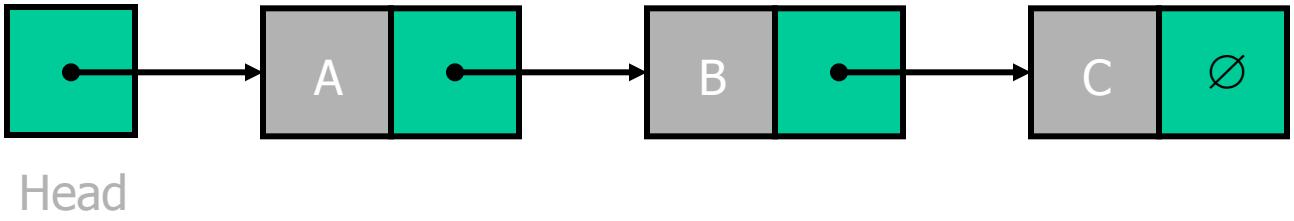
Types of Data Structures

Queue:

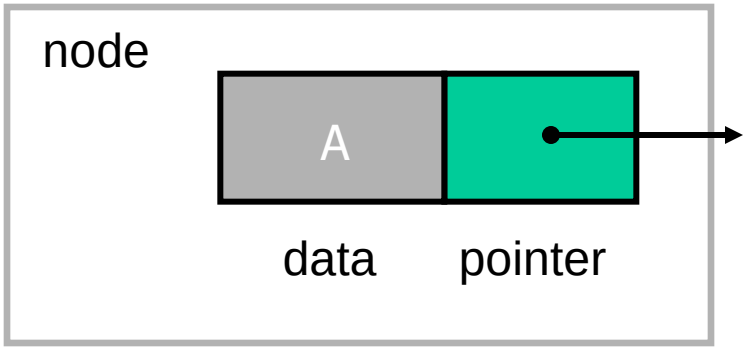
- Stores a set of elements in a particular order
- Queue principle: FIRST IN FIRST OUT
- = FIFO
- It means: the first element inserted is the first one to be removed
- Real life examples
  - Waiting in line
  - Waiting on hold for tech support
- Applications related to Computer Science
  - Threads
  - Job scheduling (e.g. Round-Robin algorithm for CPU allocation)

Types of Data Structures

Linked List:



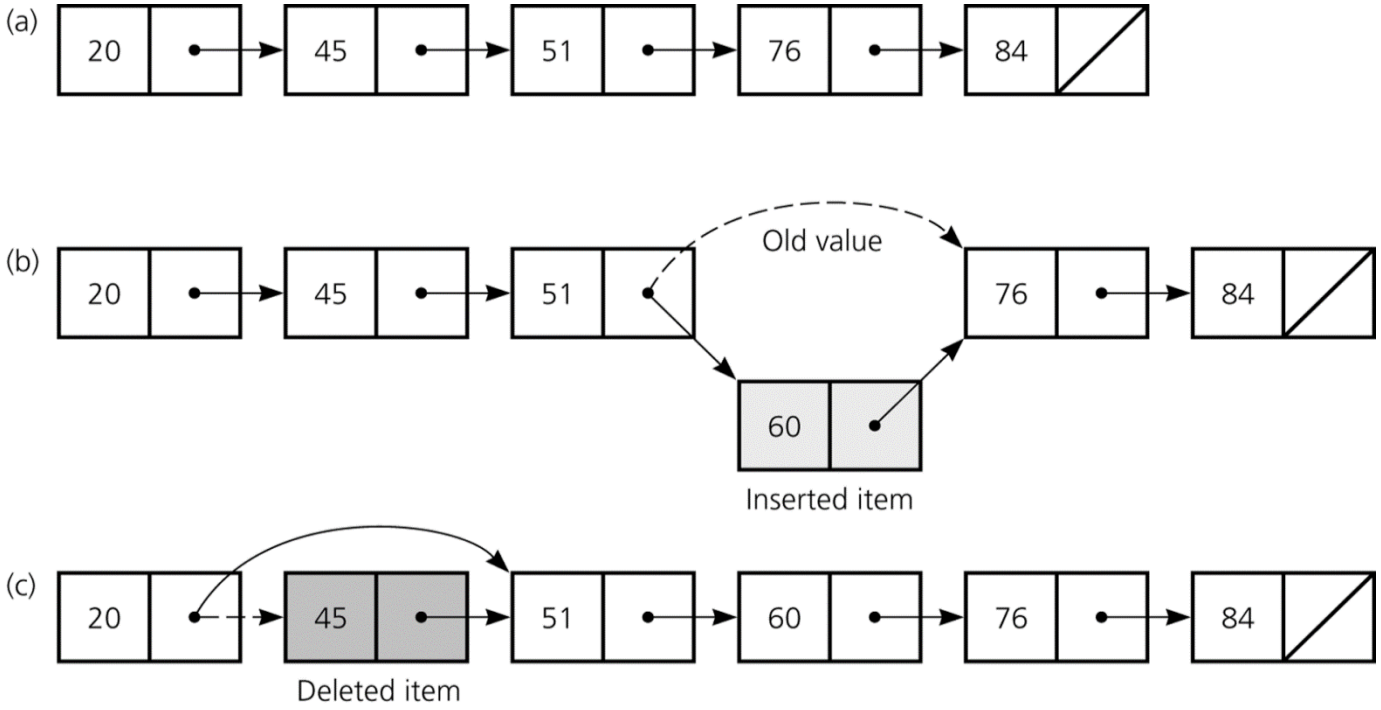
- A *linked list* is a series of connected *nodes*
- Each node contains at least
  - A piece of data (any type)
  - Pointer to the next node in the list
- *Head*: pointer to the first node
- The last node points to NULL



Types of Data Structures

Linked List:

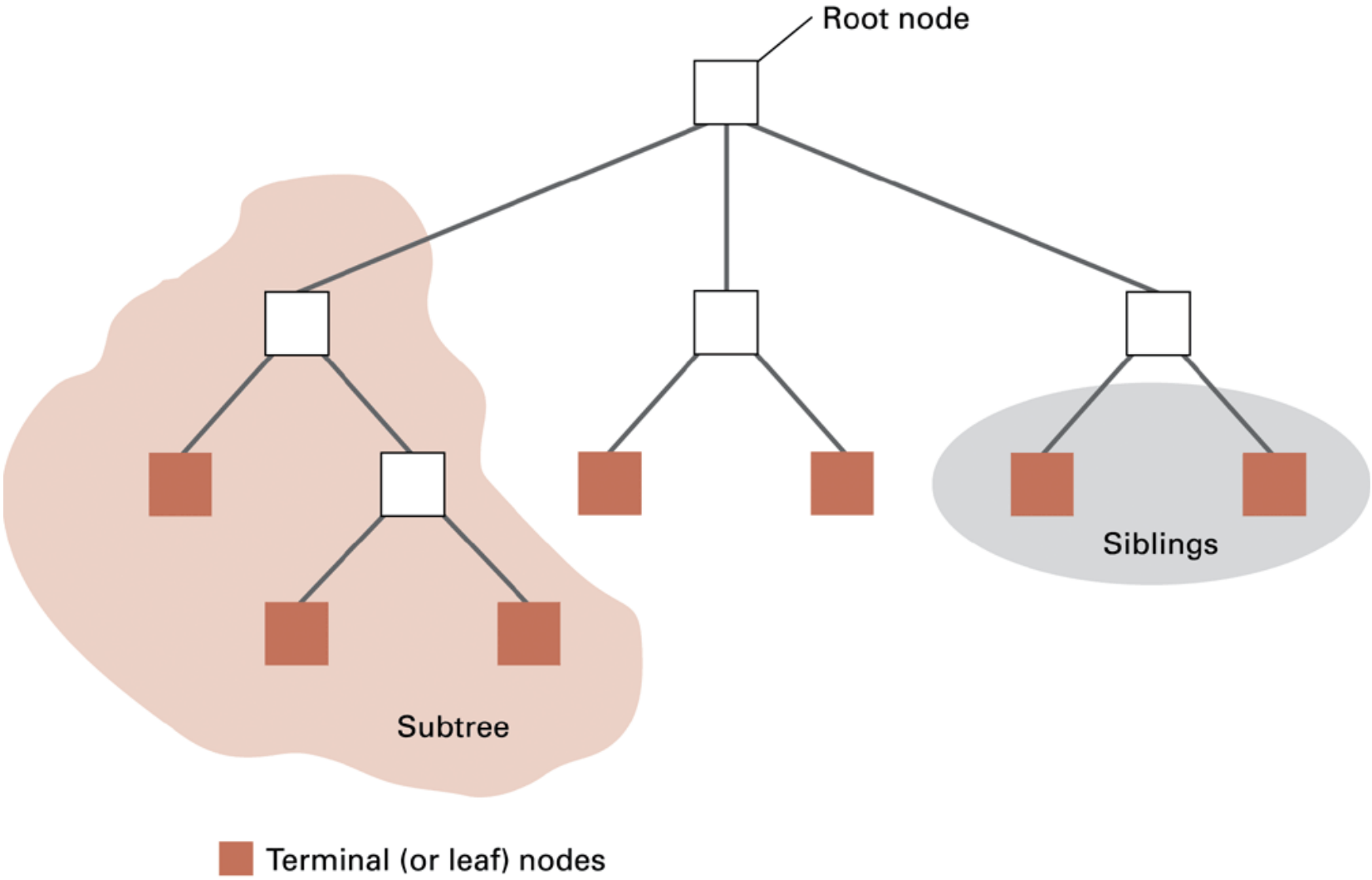
- Linear data structure.
- Primarily used when the elements are to stored at non-contiguous locations.
- Linked list is able to grow in size as needed.
- Does not require the shifting of items during insertions and deletions.



Terminology for a Tree

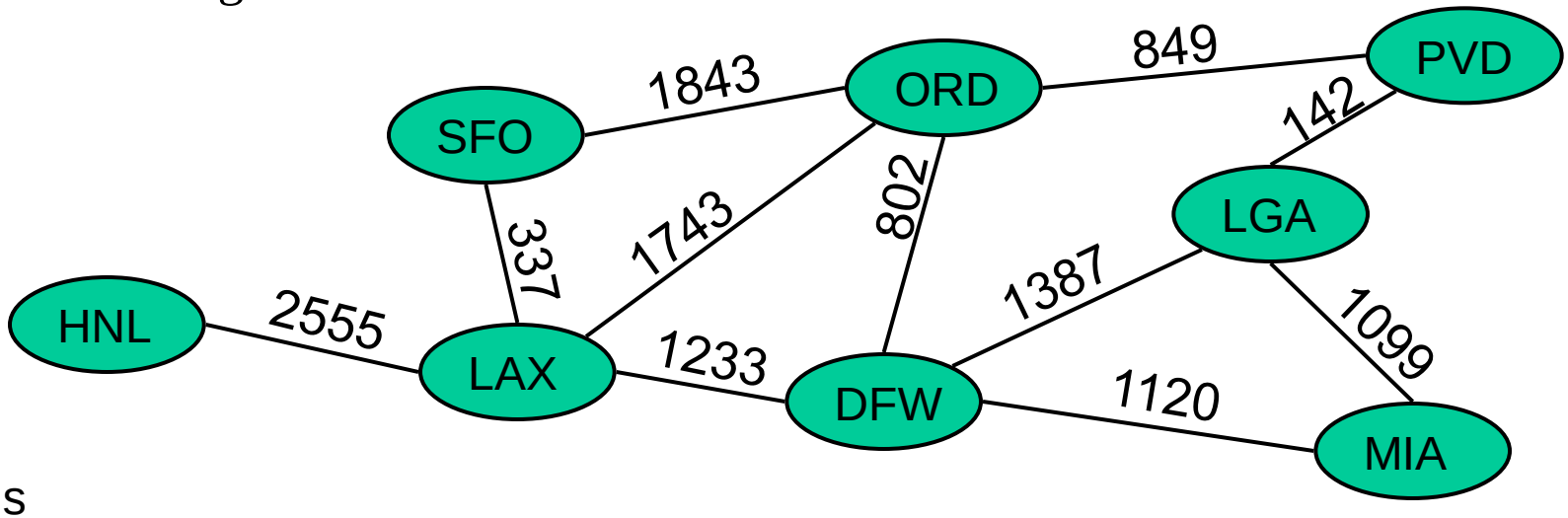
- **Tree:** A collection of data whose entries have a hierarchical organization
- **Node:** An entry in a tree
- **Root node:** The node at the top
- **Terminal** or **leaf node:** A node at the bottom
- **Parent:** The node immediately above a specified node
- **Child:** A node immediately below a specified node
- **Ancestor:** Parent, parent of parent, etc.
- **Descendent:** Child, child of child, etc.
- **Siblings:** Nodes sharing a common parent
- **Binary tree:** A tree in which every node has at most two children
- **Depth:** The number of nodes in longest path from root to leaf

Tree Terminology



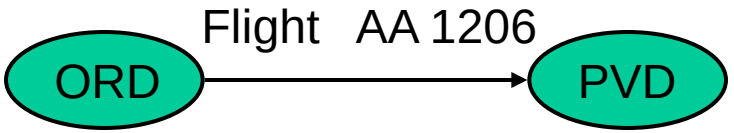
Graph

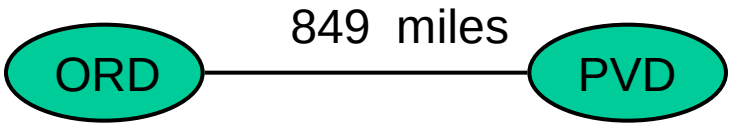
- A graph is a pair (***V, E***), where
  - ***V*** is a set of nodes, called vertices
  - ***E*** is a collection of pairs of vertices, called edges
  - Vertices and edges are positions and store elements
- Example:
  - A vertex represents an airport and stores the three-letter airport code
  - An edge represents a flight route between two airports and stores the mileage of the route



Edge Types

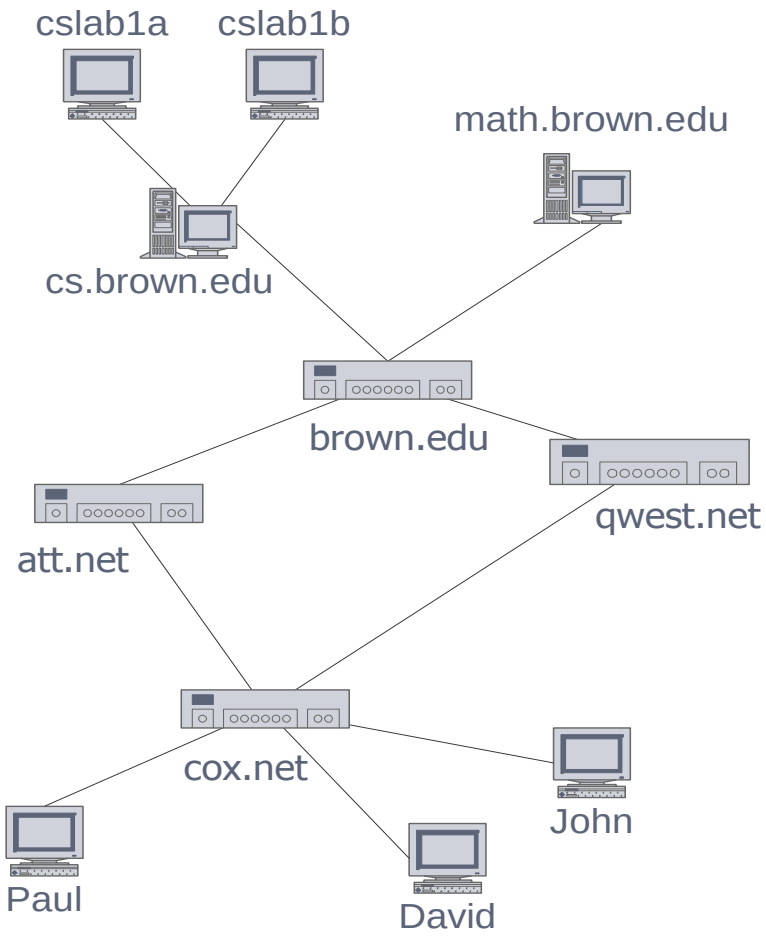
- Directed edge
  - ordered pair of vertices (***u***,***v***)
  - first vertex ***u*** is the origin
  - second vertex ***v*** is the destination
  - e.g., a flight
- Undirected edge
  - unordered pair of vertices (***u***,***v***)
  - e.g., a flight route
- Directed graph
  - all the edges are directed
  - e.g., route network
- Undirected graph
  - all the edges are undirected
  - e.g., flight network





Applications

- Electronic circuits
  - Printed circuit board
  - Integrated circuit
- Transportation networks
  - Highway network
  - Flight network
- Computer networks
  - Local area network
  - Internet
  - Web
- Databases
  - Entity-relationship diagram



Applications of Data Structure

1. **Array** can be used for sorting elements, can perform matrix operation & can be used in **CPU scheduling.**
2. **Stack** is used in Expression evaluation, Forward and backward feature in web browsers, syntax parsing, Used in many algorithms like **Tower of Hanoi** etc.
3. **Queue** is used when a resource is shared among multiple consumers like in CPU scheduling, Disk Scheduling. It is also used in **Palindrome recognition.**

4. **Binary Search Tree** is used to implement multilevel indexing in database. It is also used in **Huffman Coding** Algorithm & to implement searching Algorithm.
5. **Tree** is used as dictionary, such as one found on a mobile **telephone for auto completion and spell-checking.**
6. **Hash Table** is used for fast data lookup - symbol table for compilers, database indexing, caches,**Unique data representation.**
7. **Graphs** are used to represent networks. Graphs are also used in **social networks like linkedIn, facebook**. For example, in facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender and locale.

Complexity Analysis
and
Time-Space trade-off

Complexity

- A measure of the performance of an algorithm
- An algorithm's performance depends on
  - *internal* factors
  - *external* factors

External Factors

- Speed of the computer on which it is run
- Quality of the compiler
- Size of the input to the algorithm

Internal Factor

The algorithm's efficiency, in terms of:

- Time required to run
- Space (memory storage) required to run

Note:
Complexity measures the *internal* factors (usually more interested in time than space)

Need for analysis

- To determine resource consumption
  - CPU time
  - Memory space
- Compare different methods for solving the same problem before actually implementing them and running the programs.
- To find an efficient algorithm

Two ways of finding complexity

- Experimental study
- Theoretical Analysis

Experimental study

- Write a program implementing the algorithm
- Run the program with inputs of varying size and composition
- Get an accurate measure of the actual running time

  Use a method like System.currentTimeMillis()
- Plot the results

Limitations of Experiments

- It is necessary to implement the algorithm, which may be difficult
- Results may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments must be used
- Experimental data though important is not sufficient

Theoretical Analysis

- Uses a high-level description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, n.
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment

Space Complexity

- The space needed by an algorithm is the sum of a fixed part and a variable part
- The fixed part includes space for
  - Instructions
  - Simple variables
  - Fixed size component variables
  - Space for constants
  - etc..

Cont……

- The variable part includes space for
  - Component variables whose size is dependant on the particular problem instance being solved
  - Recursion stack space
  - etc..

Time Complexity

- The time complexity of a problem is
  - the number of steps that it takes to solve an instance of the problem as a function of the size of the input (usually measured in bits), using the most efficient algorithm.
- The exact number of steps will depend on exactly what machine or language is being used.
- To avoid that problem, the Asymptotic notation is generally used.

Time-Space trade-off

- In computer science, a **space-time** or **time-memory trade off** is a situation where the memory use can be reduced at the cost of slower program execution (or, vice versa, the computation time can be reduced at the cost of increased memory use). As the relative costs of CPU cycles, RAM space, and hard drive space change — hard drive space has for some time been getting cheaper at a much faster rate than other components of computers-the appropriate choices for space-time tradeoffs have changed radically. Often, by exploiting a space-time tradeoff, a program can be made to run much faster.

Types of Time Space Trade-off

- **Lookup tables v. recalculation**

The most common situation is an algorithm involving a lookup table: an implementation can include the entire table, **which reduces computing time, but increases the amount of memory needed**, or it can compute table entries as needed, increasing computing time, but reducing memory requirements.

- **Compressed v. uncompressed data**

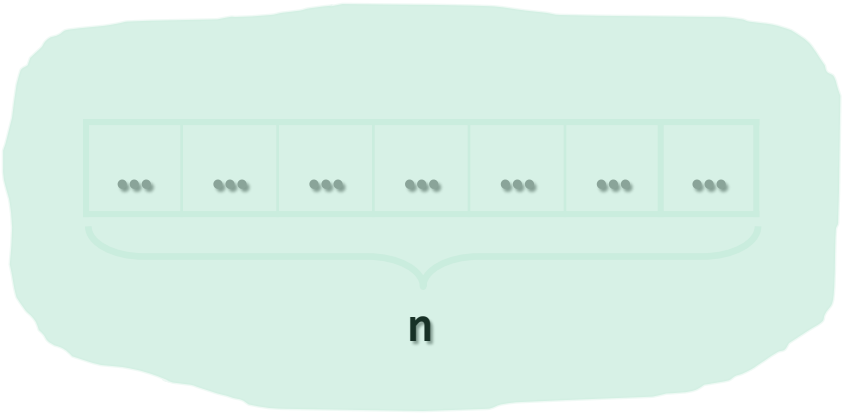
A space-time trade off can be applied to the problem of data storage. If data is stored uncompressed, it takes more space but less time than if the data were stored compressed (since compressing the data reduces the amount of space it takes, but it takes time to run the decompression algorithm). Depending on the particular instance of the problem, either way is practical.

Time Complexity

- Worst-case
  - An upper bound on the running time for any input of given size
- Average-case
  - Assume all inputs of a given size are equally likely
- Best-case
  - The lower bound on the running time

Time Complexity – Example

- Sequential search in a list of size n
  - Worst-case:
    - n comparisons
  - Best-case:
    - 1 comparison
  - Average-case:
    - n/2 comparisons



- The algorithm runs in linear time
  - Linear number of operations

Asymptotic complexity

- Asymptotic notations are mathematical tools to represent time complexity of algorithms for asymptotic analysis.
- A function giving approximate measure of efficiency of original function for large quantities of data.

$$f(n) = n^2 + 100n + \log_{10} n + 1000 \quad (2.1)$$

For small values of n, the last term, 1000, is the largest. When n equals 10, the second ($100n$) and last (1000) terms are on equal footing with the other terms making a small contribution to the function value. When n reaches the value of 100, the first and the second terms make the same contribution to the result. But when n becomes larger than 100, the contribution of the second term becomes less significant. Hence, for large values of n, due to the quadratic growth of the first term (n^2), the value of the function f depends mainly on the value of this first term, as Figure 2.1 demonstrates. Other terms can be disregarded in the long run.

FIGURE **2.1** The growth rate of all terms of function $f(n) = n^2 + 100n + \log_{10} n + 1000$.

n	f(n)	n^2		100n		$\log_{10} n$		1000	
	Value	Value	%	Value	%	Value	%	Value	%
1	1,101	1	0.1	100	9.1	0	0.0	1000	90.82
10	2,101	100	4.76	1,000	47.6	1	0.05	1000	47.62
100	21002	10,000	47.6	10,000	47.6	2	0.991	1000	4.76
1,000	1,101,003	1,000,000	90.8	100,000	9.1	3	0.0003	1000	0.09
10,000	101,001,004	100,000,000	99.0	1,000,000	0.99	4	0.0	1000	0.001
100,000	10,010,001,005	10,000,000,000	99.9	10,000,000	0.099	5	0.0	1000	0.00

Algorithms Complexity

- Algorithm complexity is rough estimation of the number of steps performed by given computation depending on the size of the input data
  - Measured through asymptotic notation
    - $O(g)$ where g is a function of the input data size
  - Examples:
    - Linear complexity $O(n)$ – all elements are processed once (or constant number of times)
    - Quadratic complexity $O(n^2)$ – each of the elements is processed n times

Comparison of the common complexities

n	Constant O(1)	Logarithmic O(log n)	Linear O(n)	Linear Logarithmic O(n log n)	Quadractic $O(n^2)$	Cubic $O(n^3)$
1	1	1	1	1	1	1
2	1	1	2	2	4	8
4	1	2	4	8	16	64
8	1	3	8	24	64	512
16	1	4	16	64	256	4,096
1,024	1	10	1,024	10,240	1,048,576	1,073,741,824