Data Structure

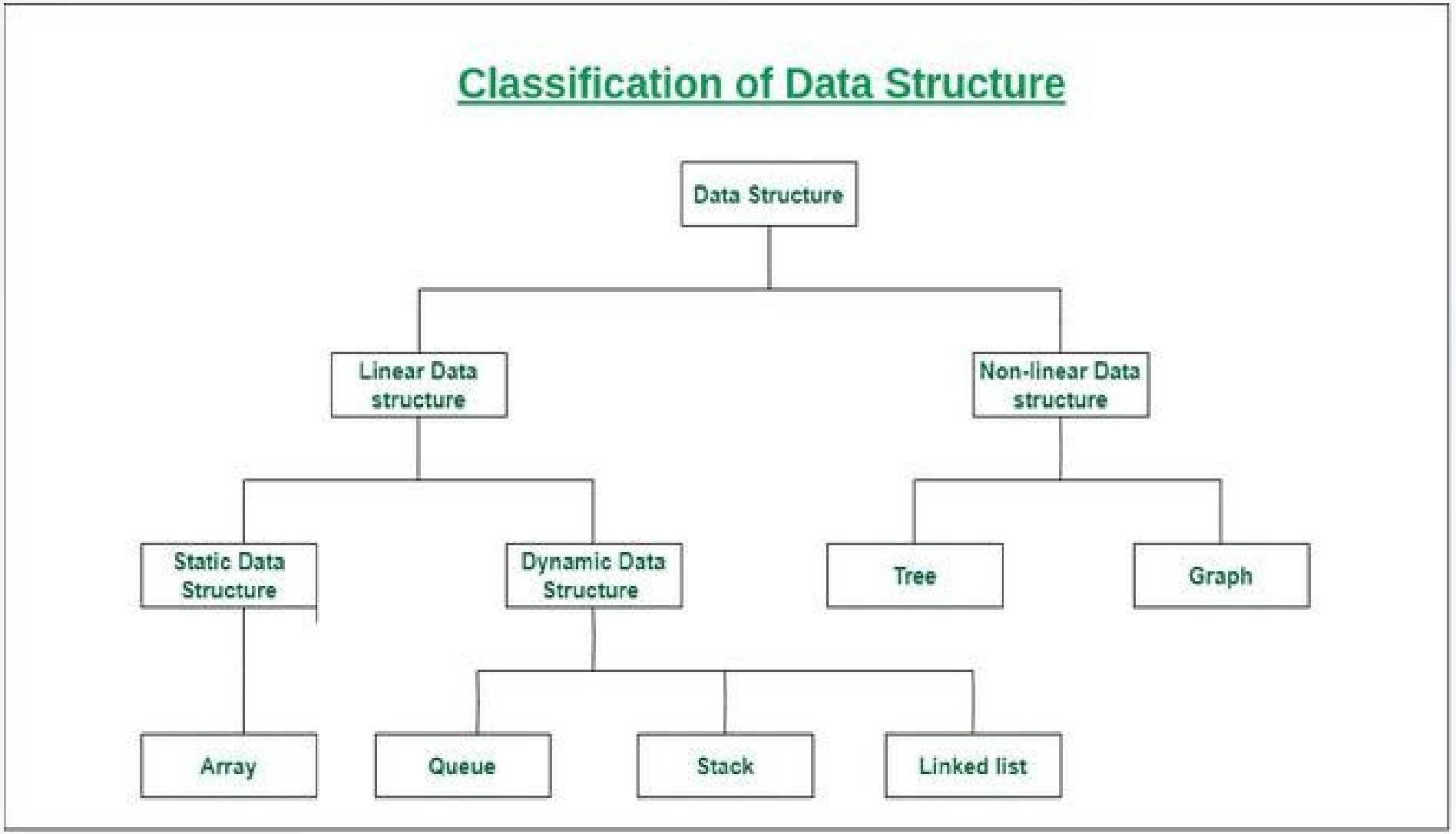
Unit 1

A.M. Joshi

What is Data Structure ?

- A data structure is a storage that is used to store and organize data.
- It is a way of arranging data on a computer so that it can be accessed and updated efficiently.
- A data structure is not only used for organizing the data.
- It is also used for processing, retrieving, and storing data.
- There are different basic and advanced types of data structures that are used in almost every program or software system that has been developed.

Types of Data Structure



Linear data structure:

- Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, is called a linear data structure.
- Examples of linear data structures are array, stack, queue, linked list, etc.

- **Static data structure:** Static data structure has a fixed memory size. It is easier to access the elements in a static data structure.
- An example of this data structure is an array.

- **Dynamic data structure:** In dynamic data structure, the size is not fixed. It can be randomly updated during the runtime which may be considered efficient concerning the memory (space) complexity of the code.
- Examples of this data structure are queue, stack, etc.

Non-linear data structure:

- Data structures where data elements are not placed sequentially or linearly are called non-linear data structures.
- In a non-linear data structure, we can’t traverse all the elements in a single run only.
- Examples of non-linear data structures are trees and graphs.

Need Of Data structure :

- The structure of the data and the synthesis of the algorithm are relative to each other. Data presentation must be easy to understand to the developer, as well as the user, can make an efficient implementation of the operation.

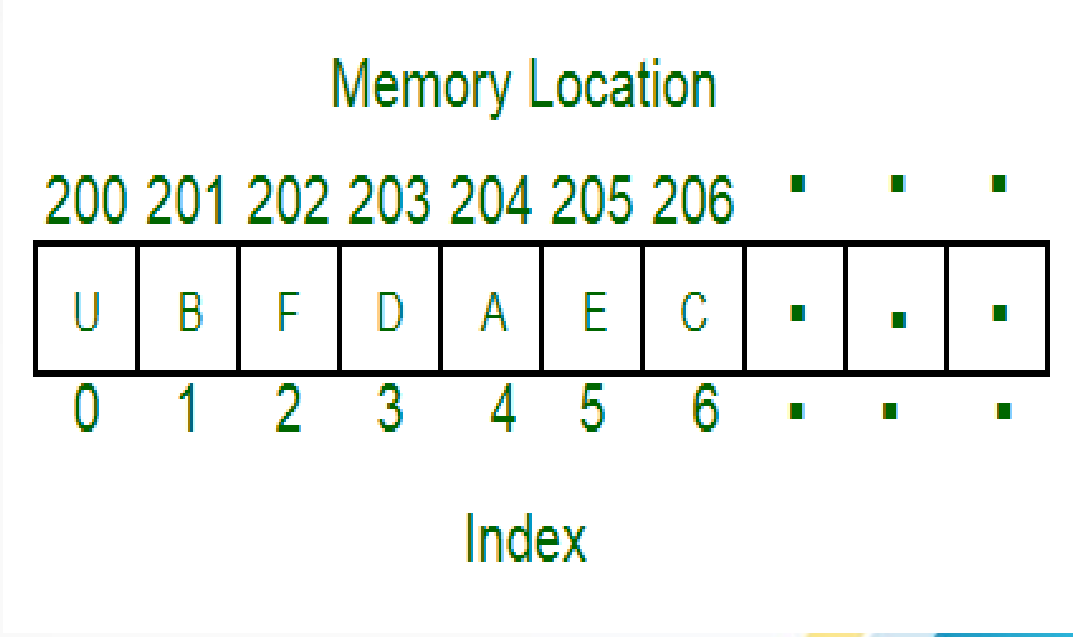
- Data structures provide an easy way of organizing, retrieving, managing, and storing data.

- Here is a list of the needs for data.

- Data structure modification is easy.
- It requires less time.
- Save storage memory space.
- Data representation is easy.
- Easy access to the large database.

Arrays:

- An array is a linear data structure and it is a collection of items stored at contiguous memory locations.
- The idea is to store multiple items of the same type together in one place.
-
- It allows the processing of a large amount of data in a relatively short period.
- The first element of the array is indexed by a subscript of 0.
- There are different operations possible in an array, like Searching, Sorting, Inserting, Traversing, Reversing, and Deleting.



Operations performed on array:

1. Initialization: An array can be initialized with values at the time of declaration or later using an assignment statement.
2. Accessing elements: Elements in an array can be accessed by their index, which starts from 0 and goes up to the size of the array minus one.
3. Searching for elements: Arrays can be searched for a specific element using linear search or binary search algorithms.
4. Sorting elements: Elements in an array can be sorted in ascending or descending order using algorithms like bubble sort, insertion sort, or quick sort.
5. Inserting elements: Elements can be inserted into an array at a specific location, but this operation can be time-consuming because it requires shifting existing elements in the array.
6. Deleting elements: Elements can be deleted from an array by shifting the elements that come after it to fill the gap.
7. Updating elements: Elements in an array can be updated or modified by assigning a new value to a specific index.
8. Traversing elements: The elements in an array can be traversed in order, visiting each element once.

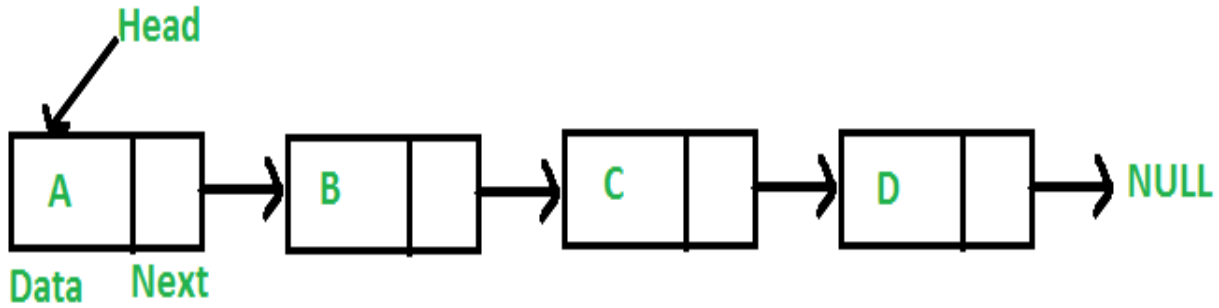
Applications of Array:

- Different applications of an array are as follows:

1. An array is used in solving matrix problems.
2. Database records are also implemented by an array.
3. It helps in implementing a sorting algorithm.
4. It is also used to implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.
5. An array can be used for CPU scheduling.
6. The array is used in many management systems like a library, students, parliament, etc.

Linked list:

- A linked list is a linear data structure in which elements are not stored at contiguous memory locations.
- The elements in a linked list are linked using pointers as shown in the below image:
- Types of linked lists:
- Singly-linked list
- Doubly linked list
- Circular linked list
- Doubly circular linked list



Characteristics of a Linked list:

- A linked list has various characteristics which are as follows:

1. A linked list uses extra memory to store links.
2. During the initialization of the linked list, there is no need to know the size of the elements.
3. Linked lists are used to implement stacks, queues, graphs, etc.
4. The first node of the linked list is called the Head.
5. The next pointer of the last node always points to NULL.
6. In a linked list, insertion and deletion are possible easily.
7. Each node of the linked list consists of a pointer/link which is the address of the next node.
8. Linked lists can shrink or grow at any point in time easily.

Operations performed on Linked list:

- A linked list is a linear data structure where each node contains a value and a reference to the next node. Here are some common operations performed on linked lists:

- Initialization: A linked list can be initialized by creating a head node with a reference to the first node. Each subsequent node contains a value and a reference to the next node.

- Inserting elements: Elements can be inserted at the head, tail, or at a specific position in the linked list.

- Deleting elements: Elements can be deleted from the linked list by updating the reference of the previous node to point to the next node, effectively removing the current node from the list.

- Searching for elements: Linked lists can be searched for a specific element by starting from the head node and following the references to the next nodes until the desired element is found.

Cont...

- Updating elements: Elements in a linked list can be updated by modifying the value of a specific node.
- Traversing elements: The elements in a linked list can be traversed by starting from the head node and following the references to the next nodes until the end of the list is reached.
- Reversing a linked list: The linked list can be reversed by updating the references of each node so that they point to the previous node instead of the next node.

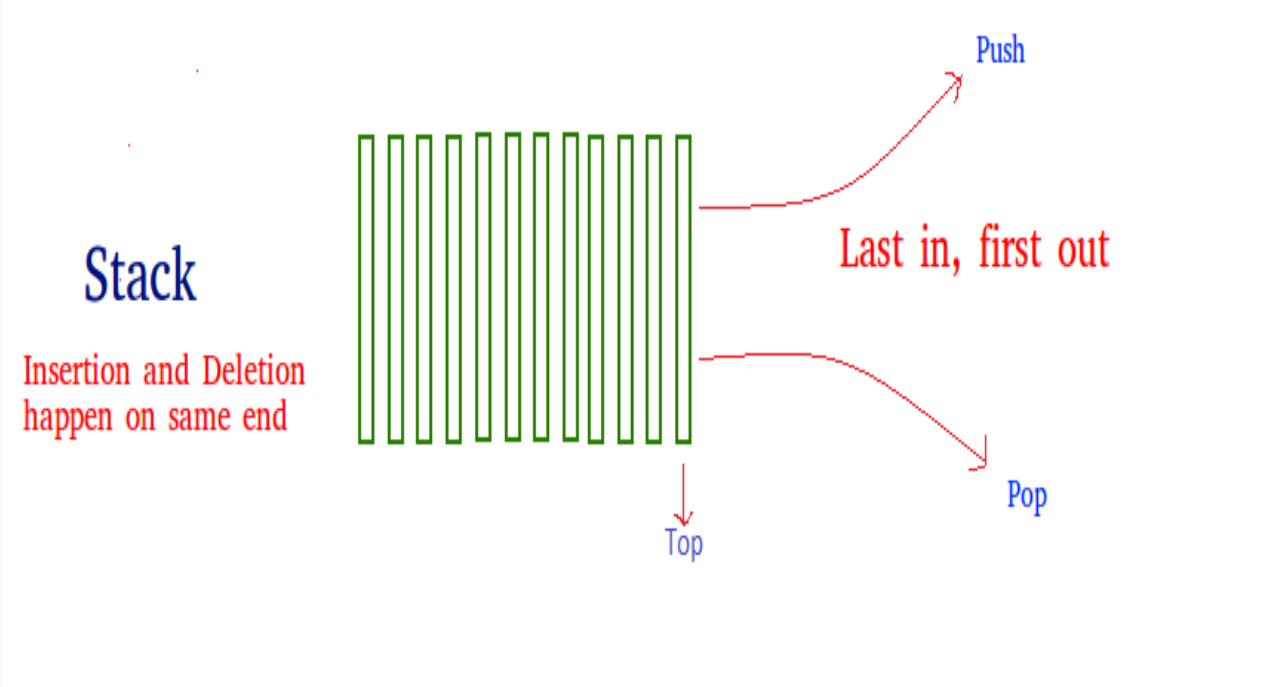
Applications of the Linked list:

- Different applications of linked lists are as follows:

1. Linked lists are used to implement stacks, queues, graphs, etc.
2. Linked lists are used to perform arithmetic operations on long integers.
3. It is used in the linked allocation of files.
4. It helps in memory management.
5. Linked lists are used to display image containers. Users can visit past, current, and next images.
6. They are used to store the history of the visited page.
7. They are used to perform undo operations.

Stack:

- Stack is a linear data structure that follows a particular order in which the operations are performed.
- The order is LIFO(Last in first out).
- Entering and retrieving data is possible from only one end.
- The entering and retrieving of data is also called push and pop operation in a stack.
- There are different operations possible in a stack like reversing a stack using recursion, Sorting, Deleting the middle element of a stack, etc.



Characteristics of a Stack:

- Stack has various different characteristics which are as follows:

1. Stack is used in many different algorithms like Tower of Hanoi, tree traversal, recursion, etc.

2. Stack is implemented through an array or linked list.

3. It follows the Last In First Out operation i.e., an element that is inserted first will pop in last and vice versa.

4. The insertion and deletion are performed at one end i.e. from the top of the stack.

5. In stack, if the allocated space for the stack is full, and still anyone attempts to add more elements, it will lead to stack overflow.

Applications of Stack:

- Different applications of Stack are as follows:

- The stack is used to convert expressions from infix to postfix.
- The stack is used to perform undo as well as redo operations in word processors.
- The stack is used in virtual machines like JVM.
- The stack is used in the media players. Useful to play the next and previous song.
- While reversing a string, the stack is used as well.
- Stack is used in memory management.

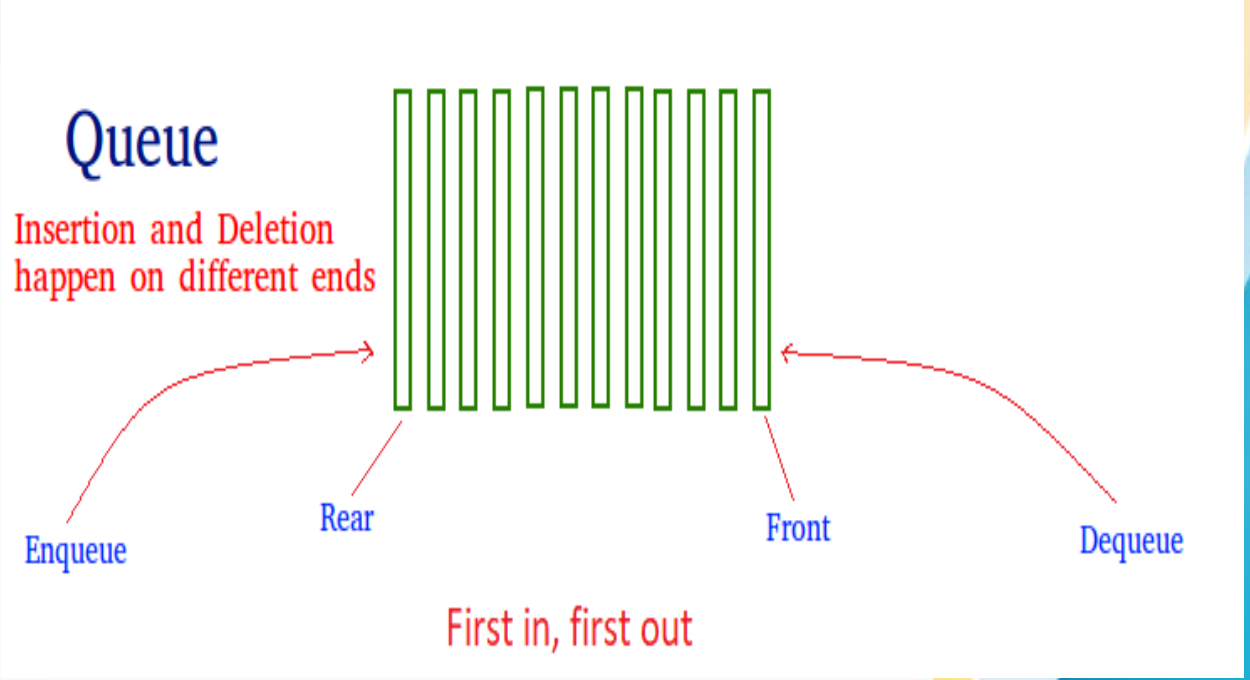
Operation performed on stack :

- A stack is a linear data structure that implements the Last-In-First-Out (LIFO) principle. Here are some common operations performed on stacks:

1. Push: Elements can be pushed onto the top of the stack, adding a new element to the top of the stack.
2. Pop: The top element can be removed from the stack by performing a pop operation, effectively removing the last element that was pushed onto the stack.
3. IsEmpty: A check can be made to determine if the stack is empty.
4. Size: The number of elements in the stack can be determined using a size operation.
5. Peek: The top element can be inspected without removing it from the stack using a peek operation.

Queue:

- Queue is a linear data structure that follows a particular order in which the operations are performed.
- The order is First In First Out(FIFO) i.e. the data item stored first will be accessed first.
- In this, entering and retrieving data is not done from only one end.
- An example of a queue is any queue of consumers for a resource where the consumer that came first is served first.



Characteristics of a Queue:

The queue has various different characteristics which are as follows:

1. The queue is a FIFO (First In First Out) structure.

2. To remove the last element of the Queue, all the elements inserted before the new element in the queue must be removed.

3. A queue is an ordered list of elements of similar data types.

Applications of Queue:

- Different applications of Queue are as follows:

1. Queue is used for handling website traffic.
2. It helps to maintain the playlist in media players.
3. It helps in serving requests on a single shared resource, like a printer, CPU task scheduling, etc.
4. It is used in the asynchronous transfer of data e.g. pipes, file IO, and sockets.
5. Queues are used for job scheduling in the operating system.
6. In social media to upload multiple photos or videos queue is used.
7. To send an e-mail queue data structure is used.

Operation performed on queue:

- A queue is a linear data structure that implements the First-In-First-Out (FIFO) principle. Here are some common operations performed on queues:

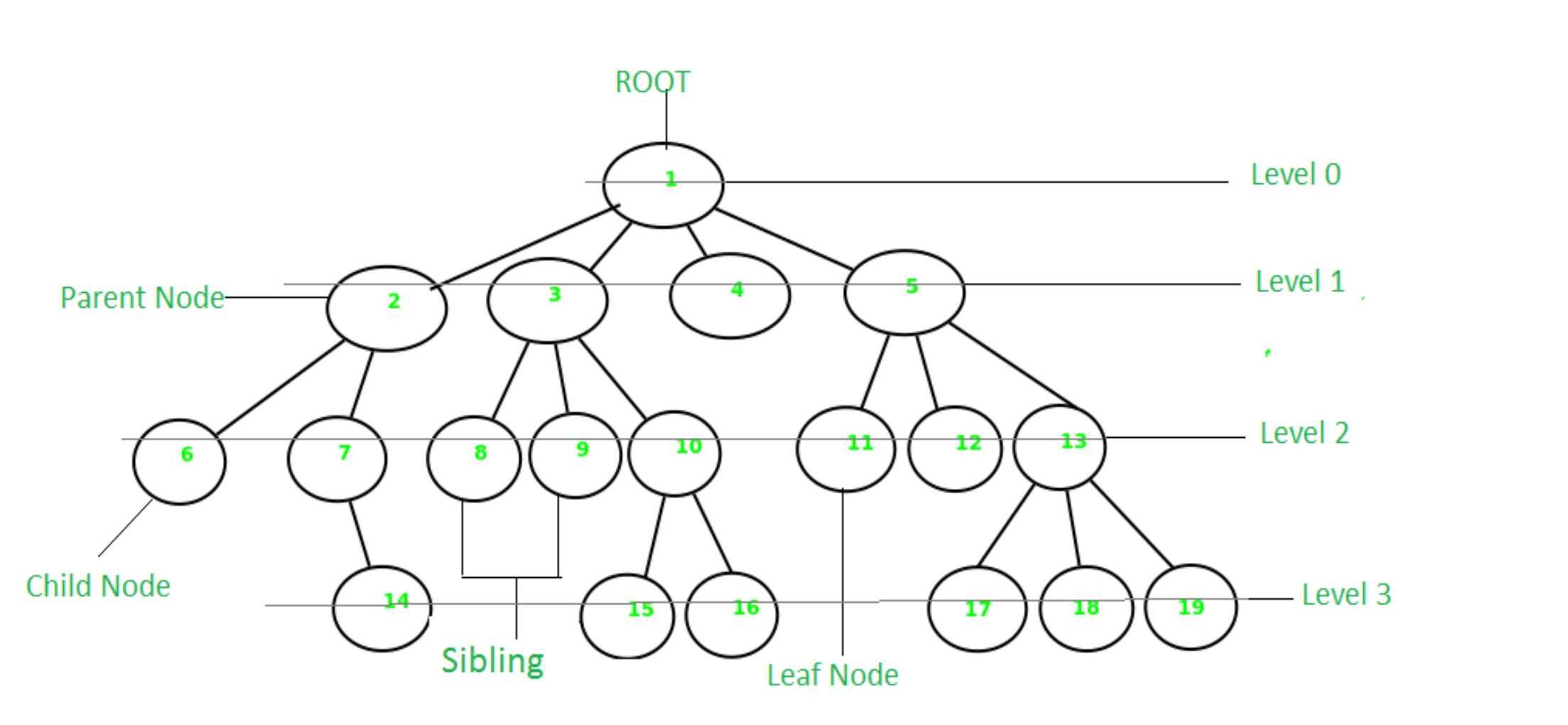
1. Enqueue: Elements can be added to the back of the queue, adding a new element to the end of the queue.
2. Dequeue: The front element can be removed from the queue by performing a dequeue operation, effectively removing the first element that was added to the queue.
3. Peek: The front element can be inspected without removing it from the queue using a peek operation.
4. IsEmpty: A check can be made to determine if the queue is empty.
5. Size: The number of elements in the queue can be determined using a size operation.

Tree:

- A tree is a non-linear and hierarchical data structure where the elements are arranged in a tree-like structure.
- In a tree, the topmost node is called the root node.
- Each node contains some data, and data can be of any type.
- It consists of a central node, structural nodes, and sub-nodes which are connected via edges.
- Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure.
- A tree has various terminologies like Node, Root, Edge, Height of a tree, Degree of a tree, etc.
- There are different types of Tree-like

- Binary Tree,
- Binary Search Tree,
- AVL Tree,
- B-Tree, etc.

Diagram Tree



Characteristics of a Tree:

- The tree has various different characteristics which are as follows:

1. A tree is also known as a Recursive data structure.

2. In a tree, the Height of the root can be defined as the longest path from the root node to the leaf node.

3. In a tree, one can also calculate the depth from the top to any node. The root node has a depth of 0.

Applications of Tree:

- Different applications of Tree are as follows:

1. Heap is a tree data structure that is implemented using arrays and used to implement priority queues.
2. B-Tree and B+ Tree are used to implement indexing in databases.
3. Syntax Tree helps in scanning, parsing, generation of code, and evaluation of arithmetic expressions in Compiler design.
4. K-D Tree is a space partitioning tree used to organize points in K-dimensional space.
5. Spanning trees are used in routers in computer networks.

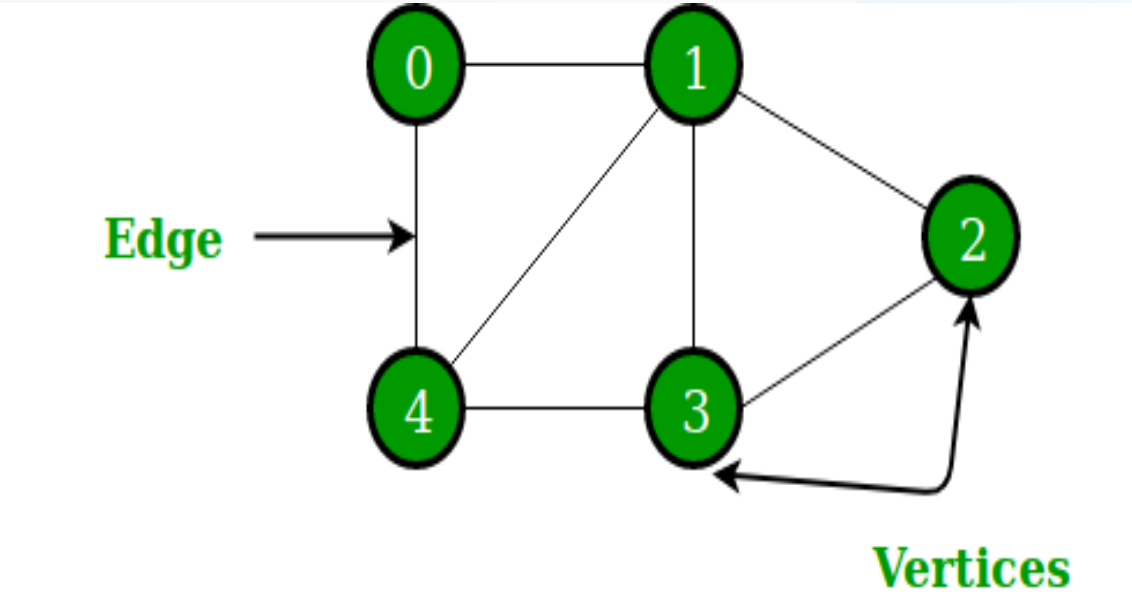
Operation performed on tree:

- A tree is a non-linear data structure that consists of nodes connected by edges. Here are some common operations performed on trees:

1. Insertion: New nodes can be added to the tree to create a new branch or to increase the height of the tree.
2. Deletion: Nodes can be removed from the tree by updating the references of the parent node to remove the reference to the current node.
3. Search: Elements can be searched for in a tree by starting from the root node and traversing the tree based on the value of the current node until the desired node is found.
4. Traversal: The elements in a tree can be traversed in several different ways, including in-order, pre-order, and post-order traversal.
5. Height: The height of the tree can be determined by counting the number of edges from the root node to the furthest leaf node.
6. Depth: The depth of a node can be determined by counting the number of edges from the root node to the current node.
7. Balancing: The tree can be balanced to ensure that the height of the tree is minimized and the distribution of nodes is as even as possible.

Graph:

- A graph is a non-linear data structure that consists of vertices (or nodes) and edges.
- It consists of a finite set of vertices and set of edges that connect a pair of nodes.
- The graph is used to solve the most challenging and complex programming problems.
- It has different terminologies which are Path, Degree, Adjacent vertices, Connected components, etc.



Applications of Graph:

- Different applications of Graphs are as follows:

1. The graph is used to represent the flow of computation.
2. It is used in modeling graphs.
3. The operating system uses Resource Allocation Graph.
4. Also used in the World Wide Web where the web pages represent the nodes.

Operation performed on Graph:

- A graph is a non-linear data structure consisting of nodes and edges. Here are some common operations performed on graphs:

- Add Vertex: New vertices can be added to the graph to represent a new node.
- Add Edge: Edges can be added between vertices to represent a relationship between nodes.
- Remove Vertex: Vertices can be removed from the graph by updating the references of adjacent vertices to remove the reference to the current vertex.
- Remove Edge: Edges can be removed by updating the references of the adjacent vertices to remove the reference to the current edge.
- Depth-First Search (DFS): A graph can be traversed using a depth-first search by visiting the vertices in a depth-first manner.

Cont...

- Breadth-First Search (BFS): A graph can be traversed using a breadth-first search by visiting the vertices in a breadth-first manner.
- Shortest Path: The shortest path between two vertices can be determined using algorithms such as Dijkstra's algorithm or A* algorithm.
- Connected Components: The connected components of a graph can be determined by finding sets of vertices that are connected to each other but not to any other vertices in the graph.
- Cycle Detection: Cycles in a graph can be detected by checking for back edges during a depth-first search.

Advantages of data structure:

1. Improved data organization and storage efficiency.
2. Faster data retrieval and manipulation.
3. Facilitates the design of algorithms for solving complex problems.
4. Eases the task of updating and maintaining the data.
5. Provides a better understanding of the relationships between data elements.

Disadvantage of Data Structure:

1. Increased computational and memory overhead.
2. Difficulty in designing and implementing complex data structures.
3. Limited scalability and flexibility.
4. Complexity in debugging and testing.
5. Difficulty in modifying existing data structures.

Algorithm

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

Complexities

- Complexity of an algorithm is a measure of the amount of time and or space required by an algorithm for an input of a given size (n).

- The complexity of an algorithm can be divided into two types:

1. Time Complexity
2. Space Complexity

Time Complexity:

- The Time complexity of an algorithm is basically the running time of a program as a function.
- Time complexity is defined as the amount of time taken by an algorithm to run, as a function of the length of the input.
- It measures the time taken to execute each statement of code in an algorithm.
- In other word the number of machine instruction which a program executes is called its Time Complexity.
- This no. is promarily dependent on the size of the program's input and the algorithm used.

Space Complexity

- The space complexity of an algorithm is the amount of computer memory that is required during the program execution as a function of the input size.
- Space complexity is a parallel concept to time complexity.
- If we need to create an array of size n, this will require O(n) space.
- If we create a two-dimensional array of size n*n, this will require O(n2) space
- Generally the space needed by a program depends on the following two parts:

1. Fixed Part: It includes the space needed for storing instructions, constant variables, and structured variables like (arrays and structures)
2. Variable part: It includes the space needed for recursion stack and for structured variables that are allocated space dynamically during the runtime of a program

Step count Method

- The step count method is one of the methods to analyze the Time complexity of an algorithm.
- In this method, we count the number of times each instruction is executed.
- Based on that we will calculate the Time Complexity.
- The step Count method is also called as Frequency Count method.
- we can calculate the step count by following below rules

1. For comment and decleration step count is 0
2. Return statement, Assignment step ccount is 1
3. Ignore lower order Exponents when higher order exponents are present.
4. Ignore constant multiplier.

EXAMPLE

```
Linearsearch(arr, n, key)
{
    i = 0;
    for(i = 0; i < n; i++)
    {
        if(arr[i] == key)
        {
            printf("Found");
        }
}
```

Where,

i = 0, is an initialization statement and takes O(1) times.

for(i = 0;i < n ; i++), is a loop and it takes O(n+1) times .

if(arr[i] == key), is a conditional statement and takes O(1) times.

printf("Found"), is a function and that takes O(0)/O(1) times.

Therefore Total Number of times it is executed is n + 4 times.

As we ignore lower exponents in time complexity total time became O(n).

Time complexity: O(n).

Order of Growth

- The order of growth of an algorithm is an approximation of the time required to run a computer program as the input size increases.
- The order of growth ignores the constant factor needed for fixed operations and focuses instead on the operation that increase proportional to input size.
- For eg: A Program with a linear order of growth generally requires double the time if the input doubles.
- The order of growth is often described using either Big-Theta or Big -O notation.

Asymptotic Notation:

- Asymptotic notation are the mathematical notations used to describe running time of an algorithm when the input tends towards a particular value or a limiting value.
- For eg: In bubble sort , when the input array is already sorted, the time taken by the algorithm is linear i.e. the best case.
- But when the input array is in reverse condition, the algorithm takes the maximum time to sort the element i.e the worst case.
- When the input array is neither sorted nor in reverse order, then it takes an average time.
- These durations are denoted using Asymptotic notations.

Cont...

- There are mainly three Asymptotic notations:

1. Big O notation
2. Omega notation
3. Theta notation

Big O notation

- Big-O notation represents the upper bound of the running time of an algorithm.
- Thus, it gives the worst-case complexity of an algorithm.

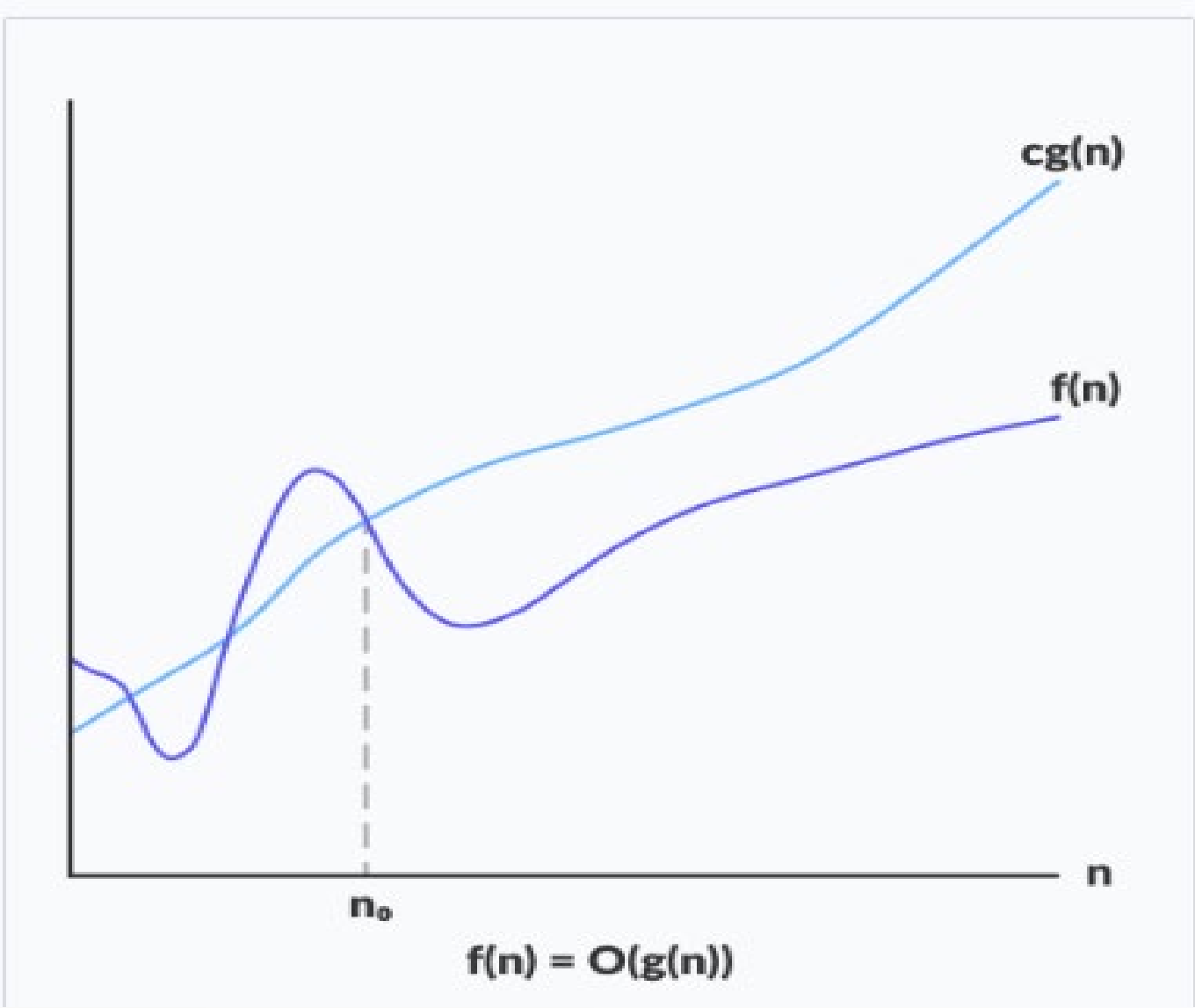
- The above expression can be described as a function f(n) belongs to the set O(g(n)) if there exists a positive constant c such that it lies between 0 and cg(n), for sufficiently large n.

- $O(g(n)) = \{ f(n)$: there exist positive constants c and n0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n0 \}$

- For any value of n, the running time of an algorithm does not cross the time provided by O(g(n)).

- Since it gives the worst-case running time of an algorithm, it is widely used to analyze an algorithm as we are always interested in the worst-case scenario.

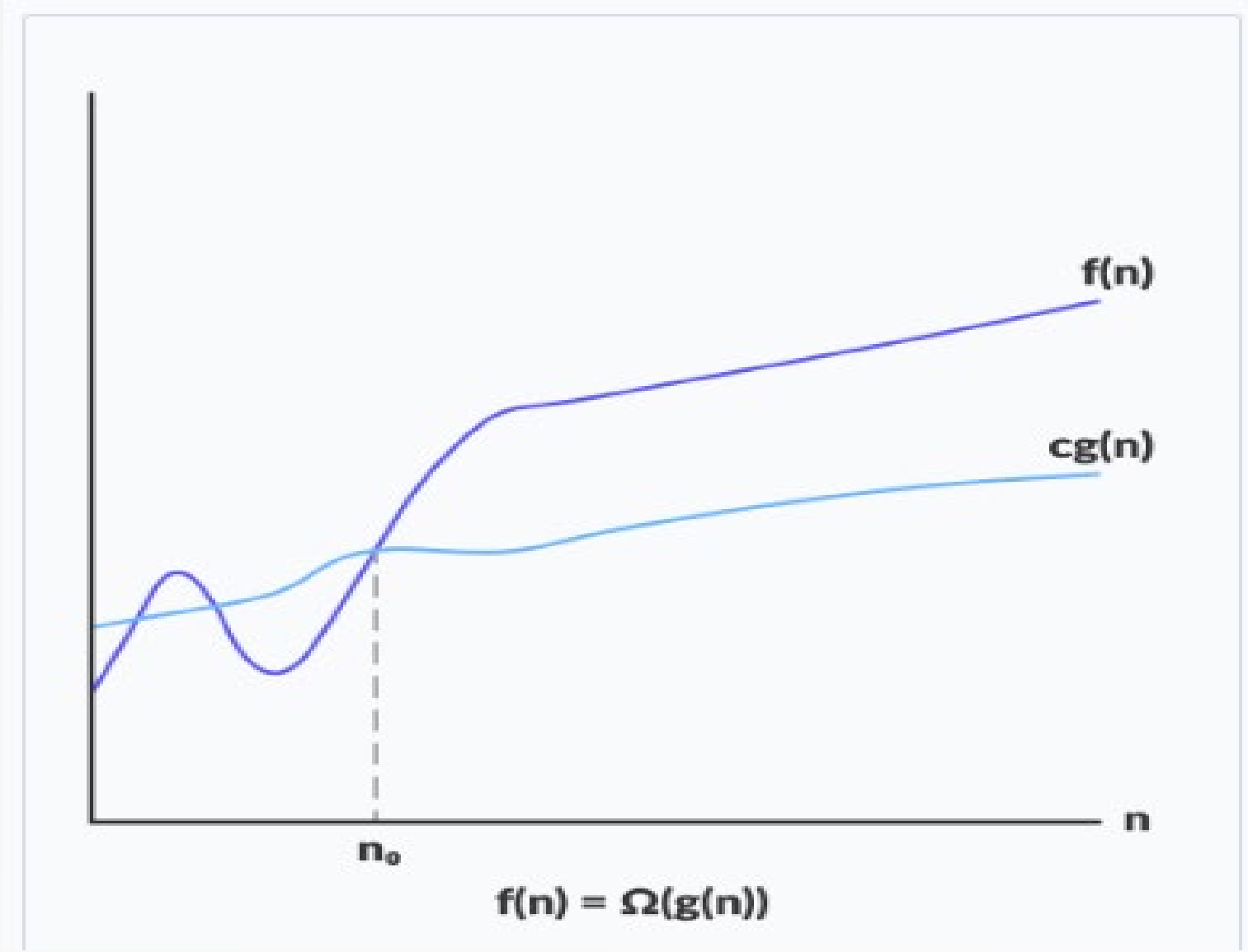
Big O Notation



Omega Notation (Ω-notation)

- Omega notation represents the lower bound of the running time of an algorithm.
- Thus, it provides the best case complexity of an algorithm.
- $\Omega(g(n)) = \{ f(n)$: there exist positive constants c and n0 ,
  such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n0 \}$
- The above expression can be described as a function f(n) belongs to the set Ω(g(n)) if there exists a positive constant c such that it lies above cg(n), for sufficiently large n.
- For any value of n, the minimum time required by the algorithm is given by Omega Ω(g(n)).

Omega Notation (Ω-notation)



Theta Notation (Θ-notation)

- Theta notation encloses the function from above and below.
- Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.

- For a function g(n), Θ(g(n)) is given by the relation:

- $\Theta(g(n)) = \{ f(n)$: there exist positive constants c1, c2 and n0
- such that $0 \leq c1g(n) \leq f(n) \leq c2g(n)$ for all $n \geq n0 \}$

- The above expression can be described as a function f(n) belongs to the set Θ(g(n)) if there exist positive constants c1 and c2 such that it can be sandwiched between c1g(n) and c2g(n), for sufficiently large n.

- If a function f(n) lies anywhere in between c1g(n) and c2g(n) for all $n \geq n0$, then f(n) is said to be asymptotically tight bound.

Theta Notation (Θ-notation)