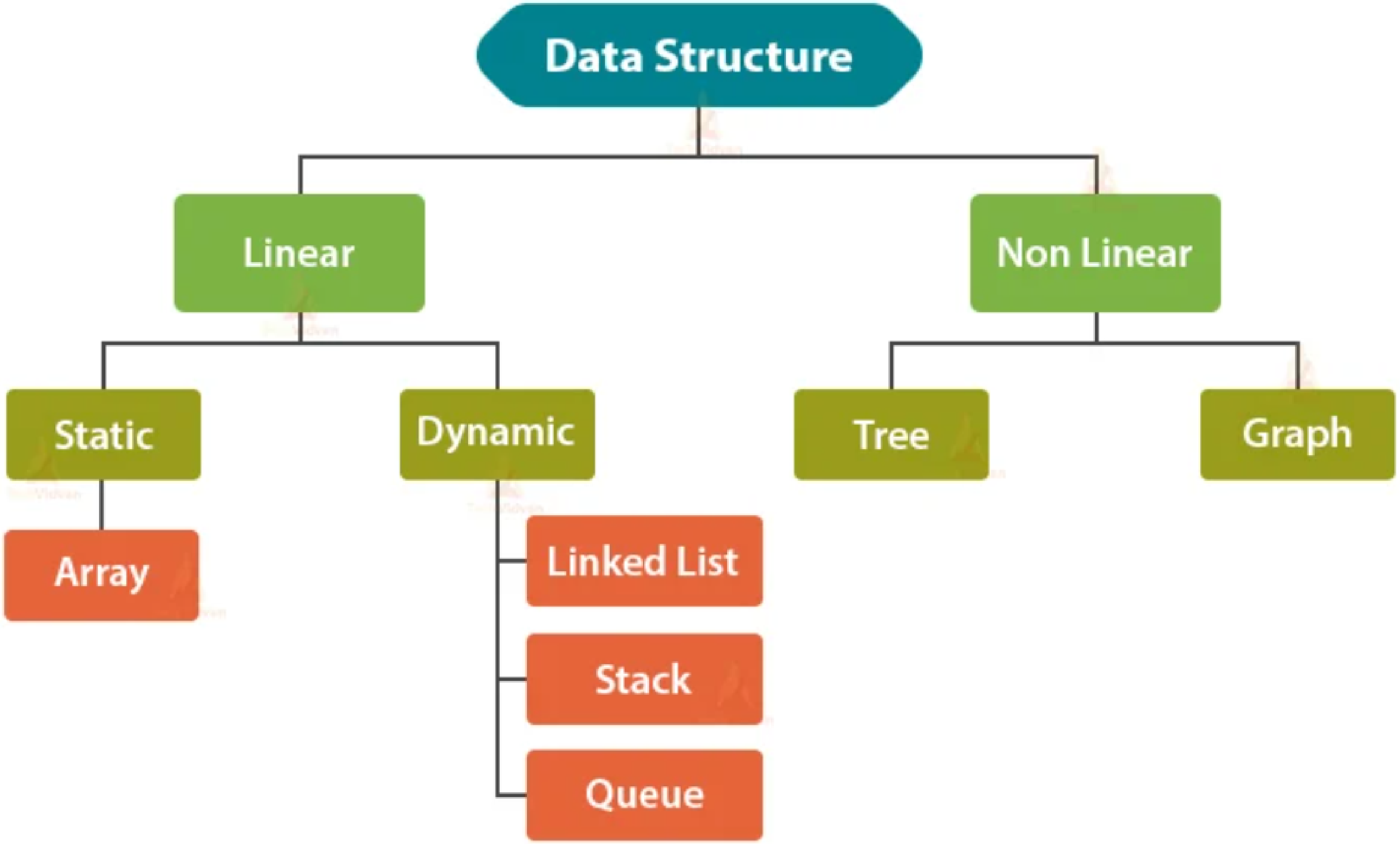
Data Structures and Algorithms

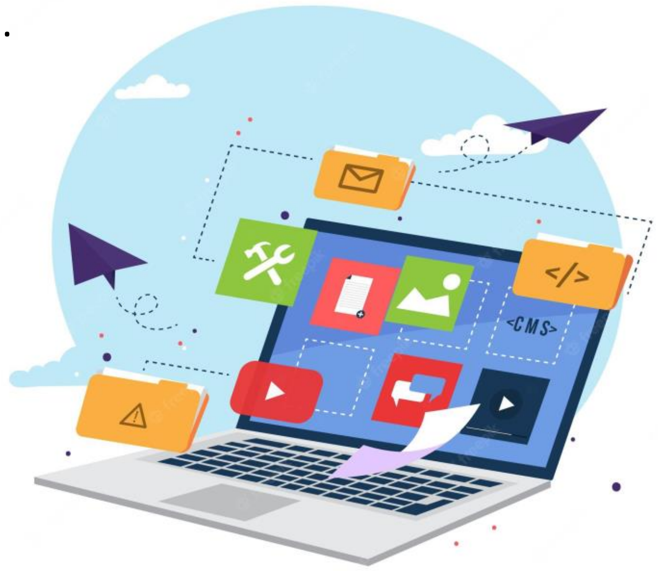
Lecture 2: Introduction **-** Basic Concepts and Notations

Data Structures:



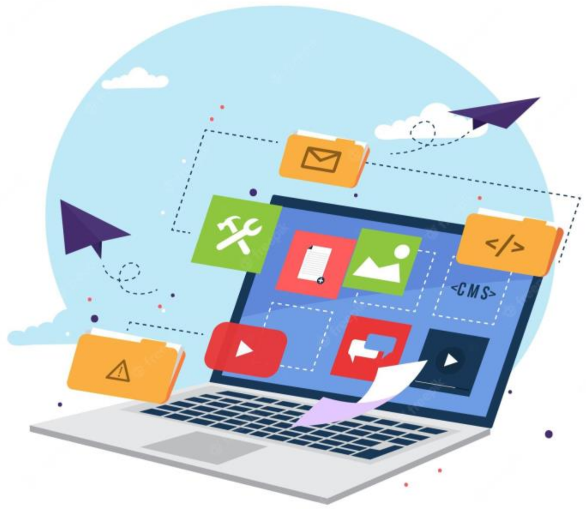
Applications of Data Structure:

- Array
  - Contacts on a cell phone.
  - Book titles in a Library Management Systems.
  - Speech processing, in which each speech signal is an array.
  - Viewing screen is also a multidimensional array of pixels.
- Strings
  - Spam email detection.
  - Plagiarism detection.
  - Search engine.
  - Digital forensic and information retrieval system
  - Spell checkers.



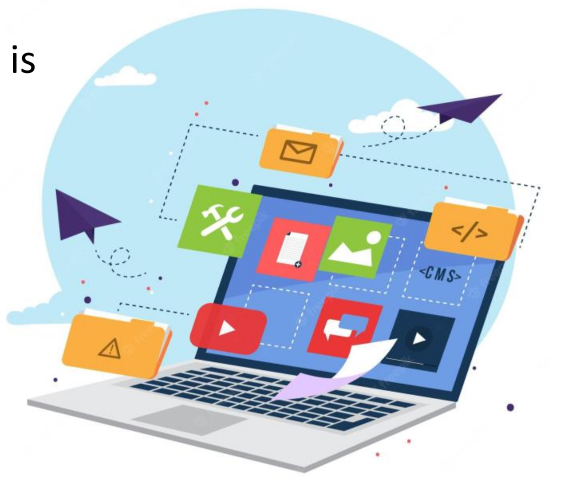
Applications of Data Structure:

- Linked list
  - Web pages can be accessed using the previous and the next URL links which are linked using a linked list.
  - The music players also use the same technique to switch between music.
  - Social media content “feeds”.
  - Syntax in the coding editor.
- Stack
  - Converting infix to postfix expressions.
  - Undo/Redo button/operation in word processors.
  - Syntaxes in languages are parsed using stacks.
  - It is used in many virtual machines like JVM.
  - Forward-backward surfing in the browser.
  - History of visited websites.



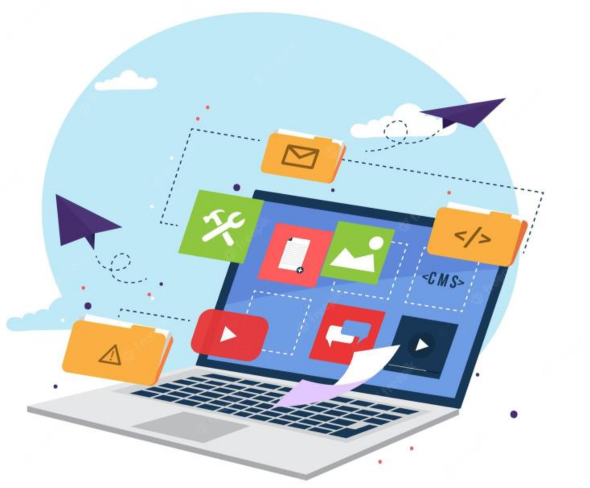
Applications of Data Structure:

- Queue
  - Operating System uses queues for job scheduling.
  - To handle congestion in the networking queue can be used.
  - Data packets in communication are arranged in queue format.
  - Sending an e-mail, it will be queued.
  - Server while responding to request
  - Uploading and downloading photos, first kept for uploading/downloading will be completed first (Not if there is threading)
  - Most internet requests and processes use queue.



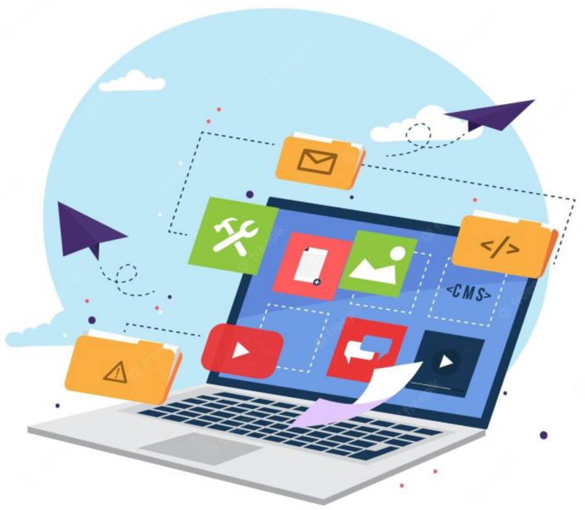
Applications of Data Structure:

- Graph
    - Facebook’s Graph API uses the structure of Graphs.
    - Google’s Knowledge Graph also has to do something with Graph.
    - Dijkstra algorithm or the shortest path first algorithm also uses graph structure to find the smallest path between the nodes of the graph.
    - The GPS navigation system also uses shortest path APIs.
    - Networking components have a huge application for graph
    - Facebook, Instagram, and all social media networking sites every user is Node
    - Data organization



Applications of Data Structure:

- Tree
  - XML Parser uses tree algorithms.
  - The decision-based algorithm is used in machine learning which works upon the algorithm of the tree.
  - Databases also use tree data structures for indexing.
  - Domain Name Server(DNS) also uses tree structures.
  - File explorer/my computer of mobile/any computer
  - BST used in computer Graphics
  - Posting questions on websites like Quora, the comments are a child of questions.



Introduction: Time space trade off

Time space trade off:

A tradeoff is a situation where one thing increases and another thing decreases. It is a way to solve a problem in:

- Either in less time and by using more space, or
- In very little space by spending a long amount of time.

Note: It is a way of solving a problem or calculation in less time by using more storage space (or memory), or by solving a problem in very little space by spending a long time.

Need of Time space trade off:

- If data is stored uncompressed, it takes more space but less time.
- Storing only the source and rendering it as an image every time the page is requested would be trading time for space. More time used but less space.

Types of Time-Space Trade-off:

1. Compressed or Uncompressed data
2. Re Rendering or Stored images
3. Smaller code or loop unrolling
4. Lookup tables or Recalculation

Complexity Analysis

The complexity of an algorithm f(N) provides the running time or storage space needed by the algorithm with respect of N as the size of input data.

Complexity Analysis:

In an algorithm and N is treated as the size of input data, the time and space implemented by the Algorithm X are the two main factors which determine the efficiency of X.

- Time Factor − The time is calculated or measured by counting the number of key operations such as comparisons in sorting algorithm.
- Space Factor − The space is calculated or measured by counting the maximum memory space required by the algorithm.

Time Complexity Analysis

1. The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.

2. Considers only the execution time of an algorithm, and not the actual execution time of the machine on which the algorithm is running on.

Example: Addition of two scalar variables.

```
Algorithm ADD SCALAR(A, B)
Input: Two scalar variables A and B
Output: variable C, which holds the addition of A and B
Operation: Addition
C <- A + B
return C
```

The of two scalar numbers requires one addition operation. the time complexity of this algorithm is constant, so $T(n) = O(1)$.

Space Complexity Analysis

1. Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.
2. It is the amount of memory needed for the completion of an algorithm.
3. The memory can be used in different forms:

    Variables (This includes the constant values and temporary values)

    Program Instruction

    Execution

Space Complexity Analysis

Example: Addition of two scalar variables.

```
Algorithm ADD SCALAR(A, B)
Input: Two scalar variables A and B
Output: variable C, which holds the addition of A and B
Operation: Addition
C <- A + B
return C
```

The addition of two scalar numbers requires one extra memory location to hold the result. Thus the space complexity of this algorithm is constant, hence $S(n) = O(1)$.

Introduction: Notations

Asymptotic Analysis

It refers to computing the running time of any operation in mathematical units of computation.

Usually, the time required by an algorithm falls under three types −

1. **Best Case** − Minimum time required for program execution.
2. **Average Case** − Average time required for program execution.
3. **Worst Case** − Maximum time required for program execution.

Asymptotic Notations

Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

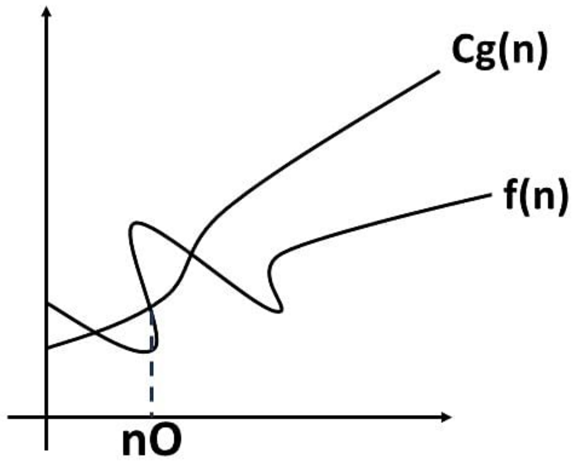
The main types are:

1. Big-O Notation
2. Omega Notation
3. Theta Notation

It compares two algorithms based on changes in their performance as the input size is increased or decreased.

Big-O Notation (O)

It is the most widely used notation for Asymptotic analysis. It specifies the upper bound of a function, i.e., the maximum time required by an algorithm or the worst-case time complexity.

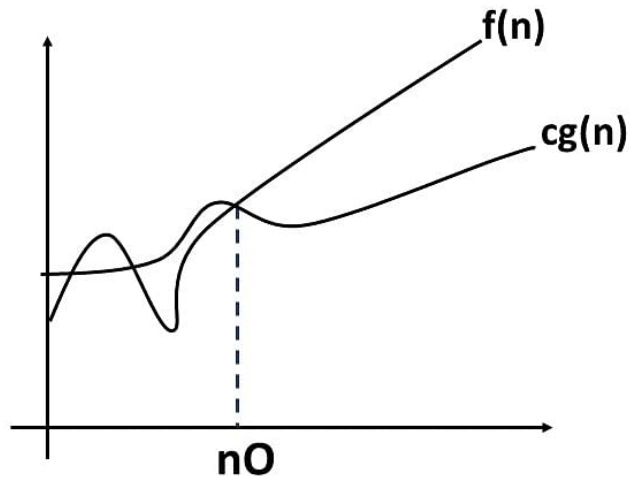


f(n) = O(g(n))

O(g(n)) = { f(n): there exist positive constants c and n0 such that 0 ≤ f(n) ≤ cg(n) for all n ≥ n0 }

Omega Notation (Ω)

Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.

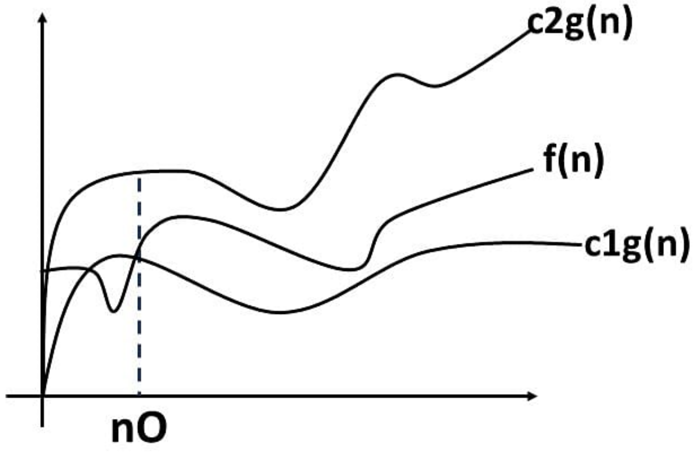


f(n) = Ω(g(n))

Ω(g(n)) = { f(n): there exist positive constants c and n0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n0$ }

Theta Notation (Θ)

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.



f(n) = θ(g(n))

*Θ (g(n)) = {f(n): there exist positive constants c1, c2 and n0 such that 0 ≤ c1 * g(n) ≤ f(n) ≤ c2 * g(n) for all n ≥ n0}*

Note: Θ(g) is a set

Recall:

- Applications of data structures (Array, Linked list, Stack, Queue, Tree and Graph)
- Time-space trade off
- Complexity analysis
- Asymptotic notations
- Various notations (Theta, Omega and Big O)