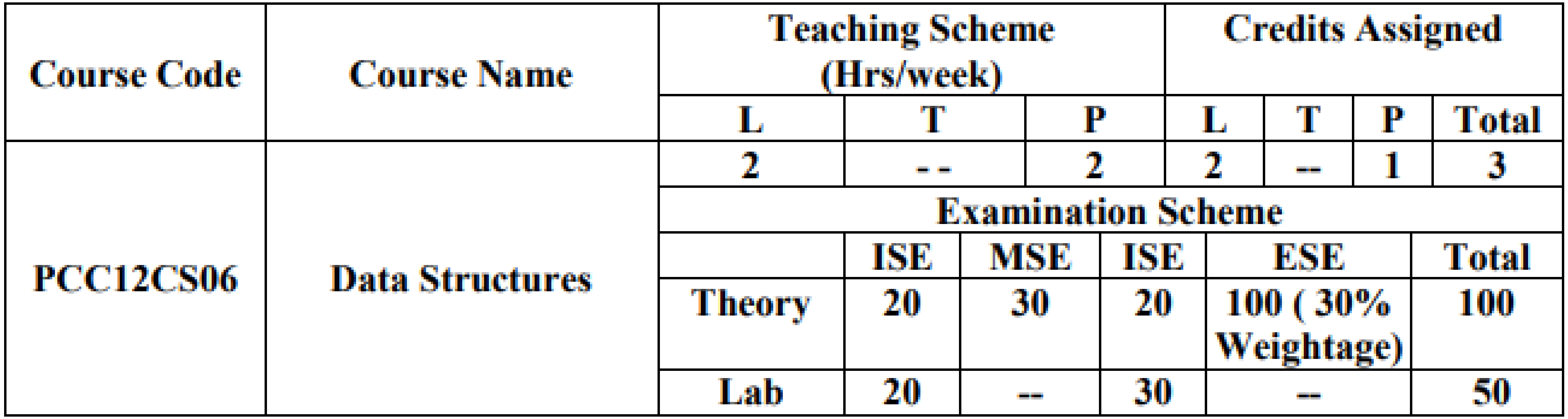
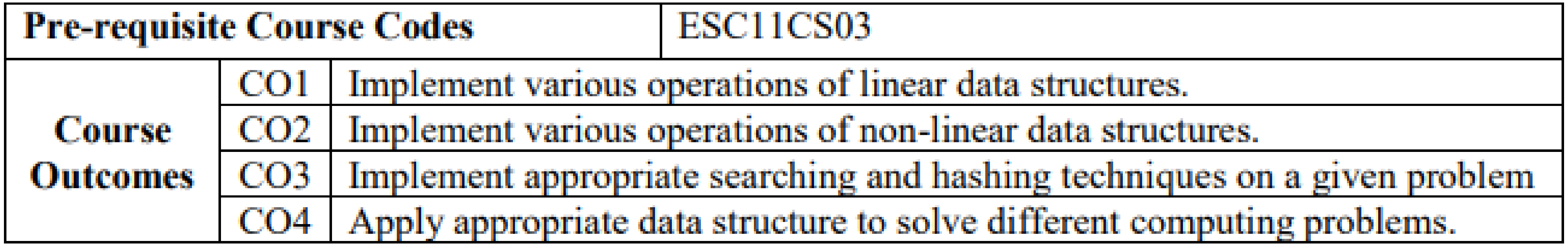
PCC12CS06:
Data Structures
And
Data Structures Lab

Teaching Scheme



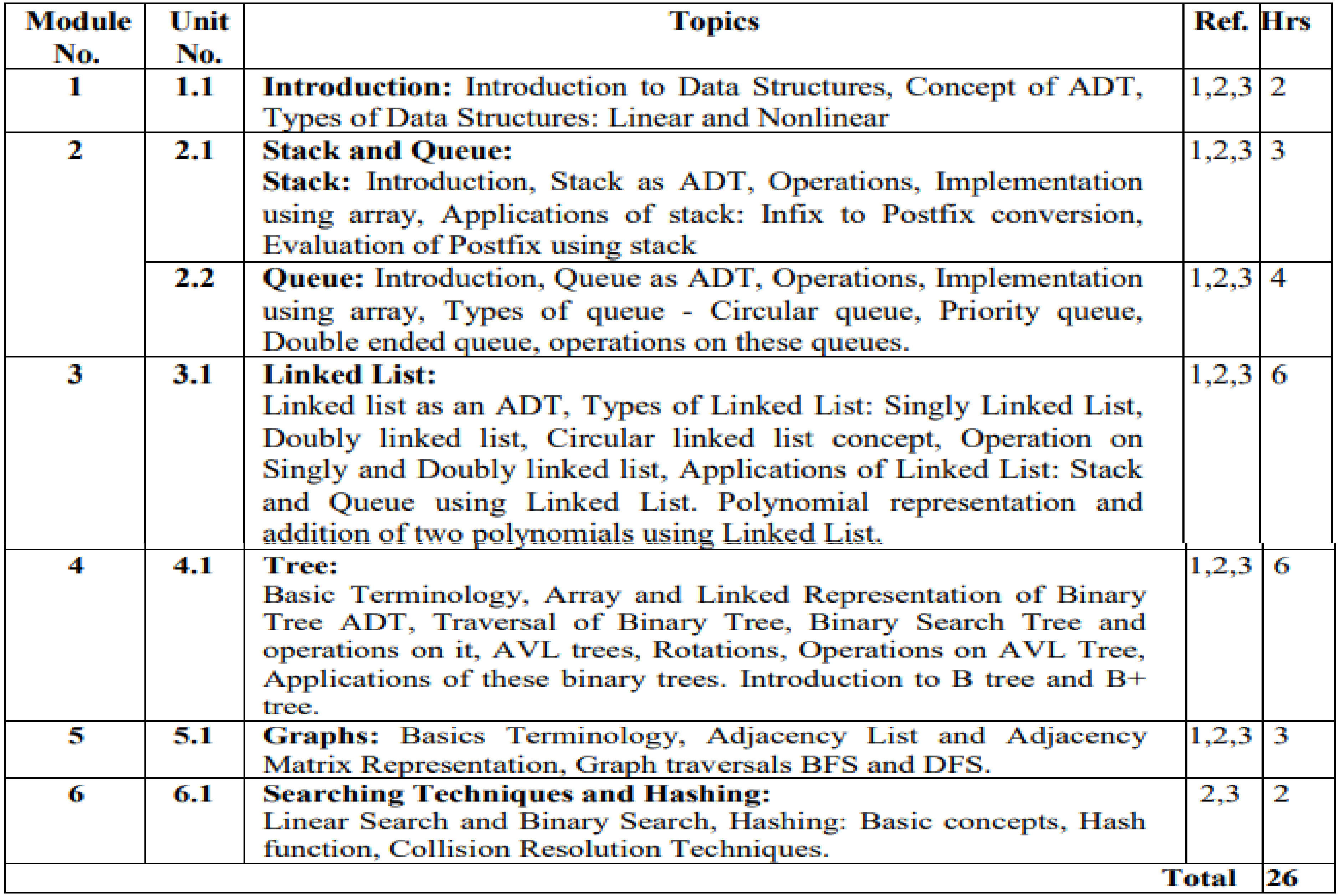
Course Code	Course Name	Teaching Scheme (Hrs/week)			Credits Assigned			
		L	T	P	L	T	P	Total
PCC12CS06	Data Structures	2	- -	2	2	--	1	3
		Examination Scheme						
			ISE	MSE	ISE	ESE		Total
		Theory	20	30	20	100 (30% Weightage)		100
		Lab	20	--	30	--		50

Course Outcomes



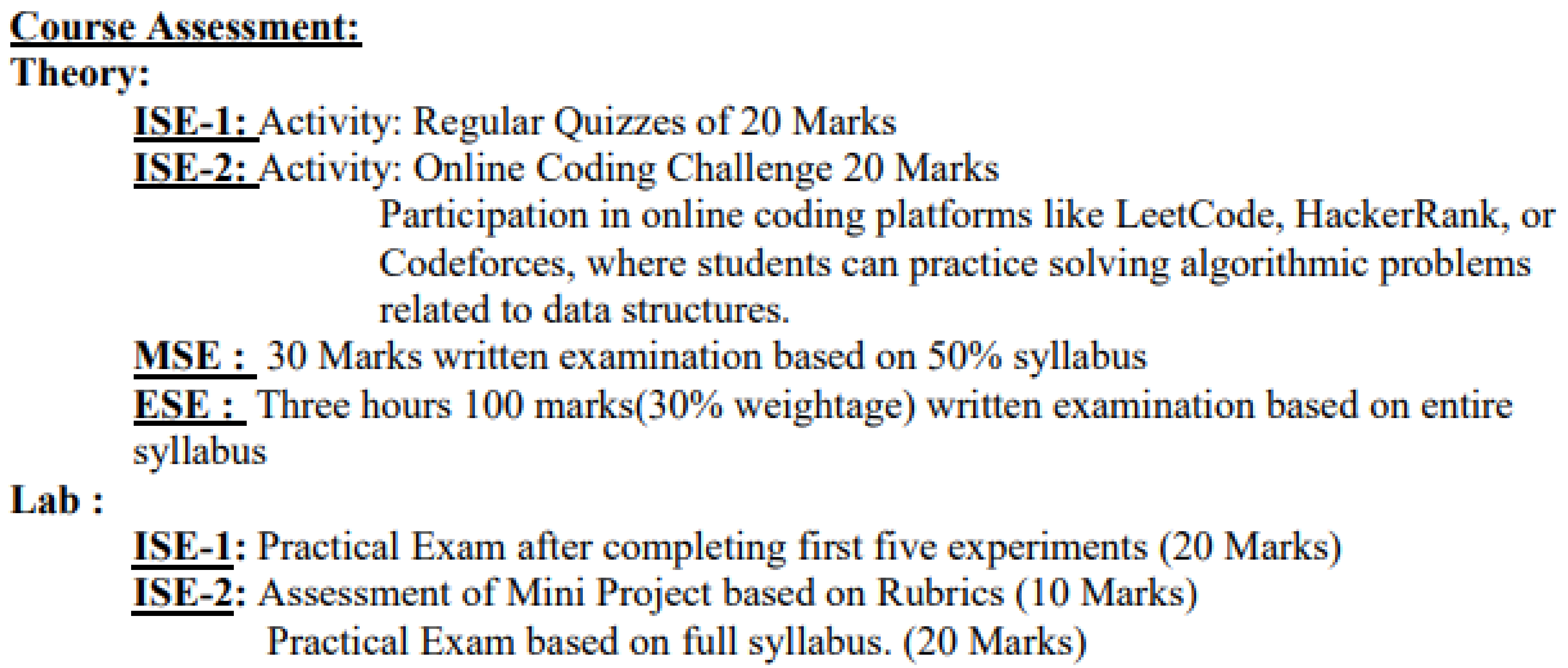
Pre-requisite Course Codes		ESC11CS03
Course Outcomes	CO1	Implement various operations of linear data structures.
	CO2	Implement various operations of non-linear data structures.
	CO3	Implement appropriate searching and hashing techniques on a given problem
	CO4	Apply appropriate data structure to solve different computing problems.

Syllabus



Module No.	Unit No.	Topics	Ref.	Hrs
1	1.1	**Introduction:** Introduction to Data Structures, Concept of ADT, Types of Data Structures: Linear and Nonlinear	1,2,3	2
2	2.1	**Stack and Queue:** **Stack:** Introduction, Stack as ADT, Operations, Implementation using array, Applications of stack: Infix to Postfix conversion, Evaluation of Postfix using stack	1,2,3	3
	2.2	**Queue:** Introduction, Queue as ADT, Operations, Implementation using array, Types of queue - Circular queue, Priority queue, Double ended queue, operations on these queues.	1,2,3	4
3	3.1	**Linked List:** Linked list as an ADT, Types of Linked List: Singly Linked List, Doubly linked list, Circular linked list concept, Operation on Singly and Doubly linked list, Applications of Linked List: Stack and Queue using Linked List. Polynomial representation and addition of two polynomials using Linked List.	1,2,3	6
4	4.1	**Tree:** Basic Terminology, Array and Linked Representation of Binary Tree ADT, Traversal of Binary Tree, Binary Search Tree and operations on it, AVL trees, Rotations, Operations on AVL Tree, Applications of these binary trees. Introduction to B tree and B+ tree.	1,2,3	6
5	5.1	**Graphs:** Basics Terminology, Adjacency List and Adjacency Matrix Representation, Graph traversals BFS and DFS.	1,2,3	3
6	6.1	**Searching Techniques and Hashing:** Linear Search and Binary Search, Hashing: Basic concepts, Hash function, Collision Resolution Techniques.	2,3	2
			Total	**26**

Assessment Tools



Course Assessment:

Theory:

- **ISE-1:** Activity: Regular Quizzes of 20 Marks
- **ISE-2:** Activity: Online Coding Challenge 20 Marks

  Participation in online coding platforms like LeetCode, HackerRank, or Codeforces, where students can practice solving algorithmic problems related to data structures.
- **MSE :** 30 Marks written examination based on 50% syllabus
- **ESE :** Three hours 100 marks(30% weightage) written examination based on entire syllabus

Lab :

- **ISE-1:** Practical Exam after completing first five experiments (20 Marks)
- **ISE-2:** Assessment of Mini Project based on Rubrics (10 Marks)

  Practical Exam based on full syllabus. (20 Marks)

Recommended Books:

1. Yedidyah Langsam, Moshe J. Augenstein, Aaron M.Tenenbaum “Data Structures using C and C++” , second edition, Pearson Publication
2. Reema Thareja, “Data Structures using C”,Third Edition, Oxford University Press.
3. Robert L. Kruse, Alexander J. Ryba, “Data Structures and Program Design in C++”, Prentice-HallIndia.
4. Data Structures and Algorithm Analysis in C by Mark Allen Weiss, second edition, Pearson Education India publication

Further Reading:

1. Michael H. Goldwasser, Michael T. Goodrich, Roberto Tamassia, “Data Structures and Algorithm in Java”, Sixth Edition 2014,Wiley publication.
2. Richard F. Gilberg &Behrouz A.Forouzan, “Data Structures: Pseudocode approach with C”, 2nd Edition, Cengage India Publication

Online Resources:

1. https://nptel.ac.in/courses/106/102/106102064/
2. https://www.coursera.org/specializations/data-structures-algorithms
3. https://visualgo.net
4. www.leetcode.com
5. www.hackerrank.com
6. www.codechef.co

Module 1:
Introduction to Data Structures

Design Good Programs

- Good program is defined as a program that
  1. **Runs correctly**
  2. **Easy to read and understand**
  3. **Easy to debug and**
  4. **Easy to modify.**
- A program should undoubtedly give correct results, but along with that it should also **run efficiently.**
- A program is said to be efficient when it executes in **minimum time and with minimum memory space.**
- In order to write efficient programs we need to **apply certain data management concepts**.
- The concept of **data management** is a complex task that includes **activities like data collection, organization of data into appropriate structures, and developing and maintaining routines for quality assurance.**

Data structure

- A data structure is basically a **group of data elements** that are **put together under one name**, and which **defines a particular way of storing and organizing data** in a computer so that it can be used efficiently.

- Data structures are **used in almost every program** or **software system**.

- Some common examples of data structures are **arrays, linked lists, queues, stacks, binary trees, and hash tables.**

- Data structures are widely applied in the following areas:

  Compiler design
  Statistical analysis package
  Numerical analysis
  Artificial intelligence
  Operating system
  DBMS
  Simulation
  Graphics

Classification of Data Structures

- Data structures are generally categorized into two classes**:**
    1. **Primitive Data Structures.**
    2. **Non-Primitive Data Structures.**

- **Primitive and Non-Primitive Data Structures**
    - **Primitive data structures:**
        - Primitive data structures are the **fundamental data types** which are supported by a programming language.
        - Some basic data types are **integer, real, character, and Boolean**.

    - **Non-primitive data structures:**
        - **Non-primitive data structures** are those data structures which are **created using primitive data structures.**
        - Examples of such data structures include **linked lists, stacks, trees, and graphs.**
        - Non-primitive data structures can further be classified into two categories: **linear and non-linear data structures**.

Linear and Non-linear Structures

- **LINEAR STRUCTURE:**
  - If the elements of a data structure are stored in a **linear or sequential order**, then it is a linear data structure.
  - Examples include **arrays, linked lists, stacks, and queues**.
  - Linear data structures can be **represented in memory** in **two** different ways.
    - One way is to have to a linear relationship between elements by means of **sequential memory locations**.
    - The other way is to have a linear relationship between elements by **means of links**.

- **NON-LINEAR STRUCTURES:**
  - If the elements **do not follow a linear sequence or arrangement** of elements and are **more complex** than primitive data structures.
  - These data structures **uses for more flexible and efficient storage, retrieval, and manipulation of data.**
  - Examples includes: **Binary Search Tree (BST),AVL Tree, Red-Black Tree, Trie,**

  B-tree, Graph, Hash Table, Heap, Quad-tree, Octree

Abstract Data Type

- Data type of a variable is the **set of values that the variable can take.**
- We have already read the basic data types in C include int, char, float, and double.
- The word ‘abstract’ in the context of data structures means **considered apart from the detailed specifications or implementation.**
- An abstract data type (ADT) is the way we look at a data structure, **focusing on what it does and ignoring how it does its job.**
- For example**, stacks and queues** are perfect examples of an ADT.
- We can implement both these ADTs using an **array or a linked list.**
- **This demonstrates the ‘abstract’ nature of stacks and queues**

Algorithms

- Algorithm is ‘a **Formally Defined Procedure For Performing Some Calculation’.**

- If a procedure is formally defined, then it can be **implemented using a formal language,** and such a language is known as a **programming language.**

- In general terms, an algorithm provides a **blueprint to write a program to solve a particular problem.**

- It is considered to be an **effective procedure for solving a problem in finite number of steps**.

- That is, a **well-defined algorithm always provides an answer and is guaranteed to terminate.**

- It is not uncommon to have multiple algorithms to tackle the same problem, but the choice of a particular algorithm must **depend on the time and space complexity** of the algorithm.

Different Approaches To Designing An Algorithm

- Two main approaches to design an algorithm—
  - Top-down approach
  - Bottom-up approach

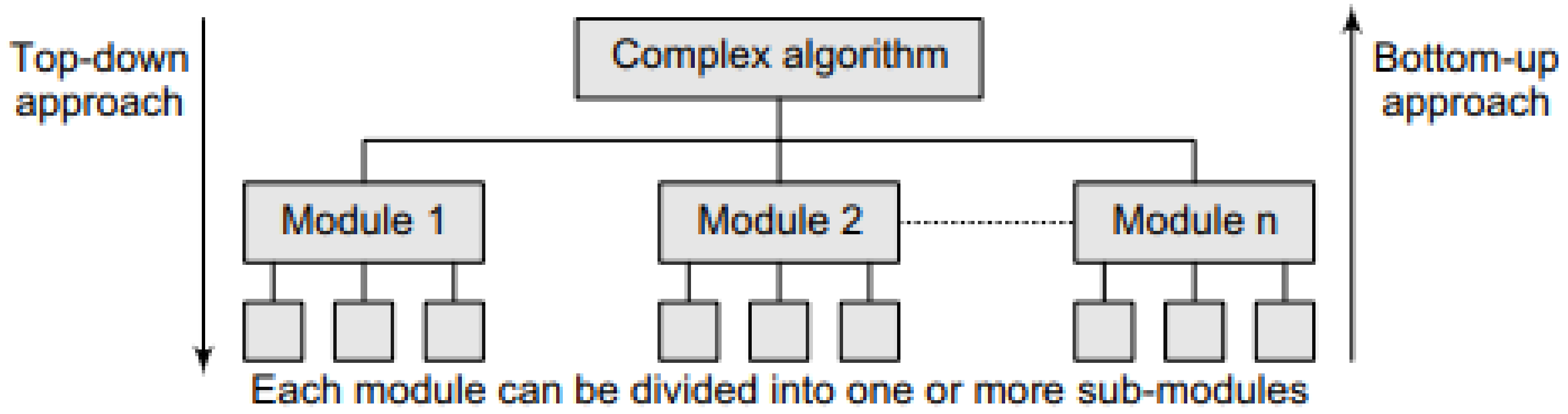


Figure 2.9 Different approaches of designing an algorithm

Top-down approach

- A top-down design approach starts by **dividing the complex algorithm into one or more modules.**

- These modules can **further be decomposed into one or more sub-modules**, and this process of decomposition is iterated until the **desired level of module complexity is achieved**.

- Top-down design method is a form of stepwise refinement where **we begin with the topmost module and incrementally add modules that it calls.**

- Therefore, in a top-down approach, we **start from an abstract design** and then at each step, this design is **refined into more concrete levels** until a level is reached that requires no further refinement.

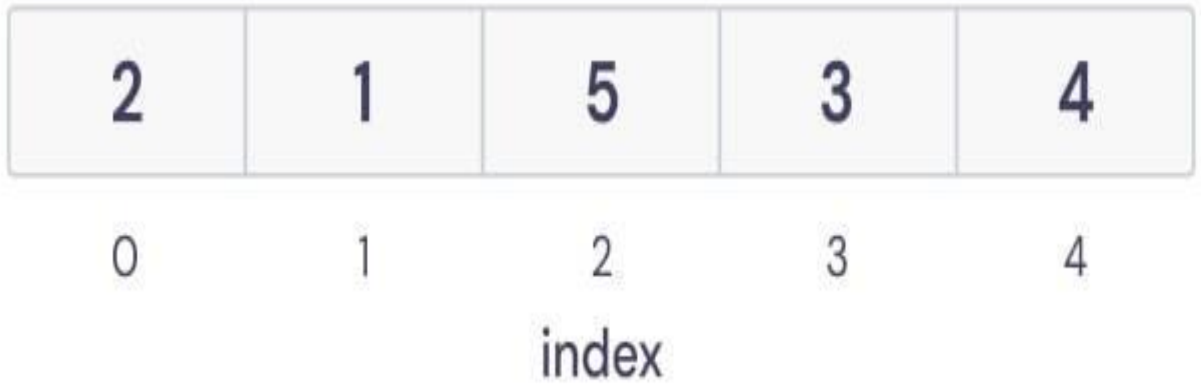
Bottom-up approach

- A bottom-up approach is just the **REVERSE** of top-down approach.
- In the bottom-up design, we start with designing the **most basic or concrete modules** and then proceed towards designing higher level modules.
- The higher level modules are implemented by using the operations performed by lower level modules.
- Thus, in this approach **sub-modules are grouped together** to **form a higher level module.**
- All the higher level modules are clubbed together to form even higher level modules.

S.No.	Top-Down Approach	Bottom-Up Approach
1.	In this approach, the problem is broken down into smaller parts.	In this approach, the smaller problems are solved.
2.	It is generally used by structured programming languages such as C, COBOL, FORTRAN, etc.	It is generally used with object oriented programming paradigm such as C++, Java, Python, etc.
3.	It is generally used with documentation of module and debugging code.	It is generally used in testing modules.
4.	It does not require communication between modules.	It requires relatively more communication between modules.
5.	It contains redundant information.	It does not contain redundant information.
6.	Decomposition approach is used here.	Composition approach is used here.
7.	The implementation depends on the programming language and platform.	Data encapsulation and data hiding is implemented in this approach.

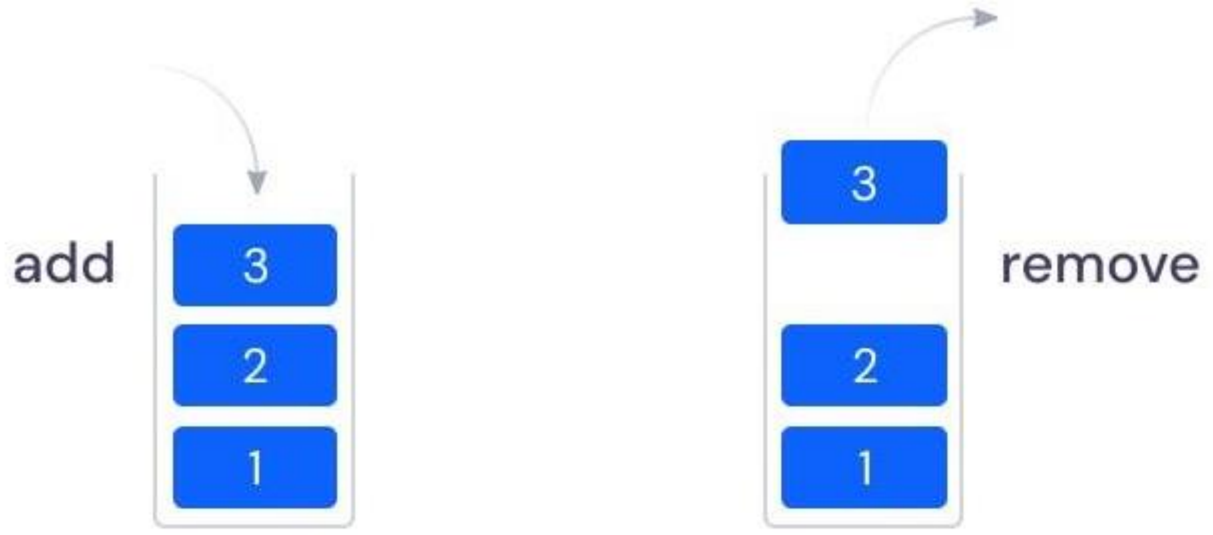
Common linear data structures:

1. **Array:**
   - An array is a data structure that stores a sequence of elements.
   - Arrays typically store data in memory, but they can also represent relationships between pieces of data.



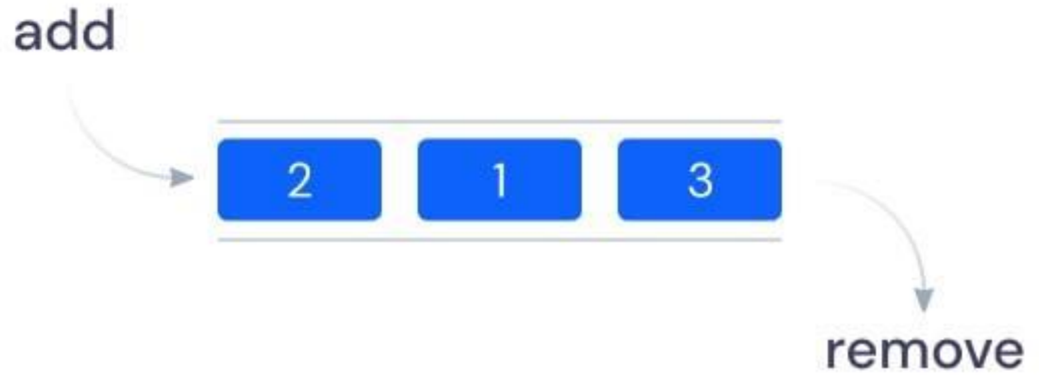
2. **Stacks:**

- A stack is a data structure that **stores a sequence of elements**, with the most recently added element at the **top of the stack**.
- A stack is a data structure that organizes several pieces of information in the order in which operations are applied to them.
- The order is Last in, First out – **LIFO,** which is the same as First in, Last out – **FILO.**



- **Queue:**
- A queue is a data structure that stores a sequence of elements, with the most **recently added element at the back of the queue**.

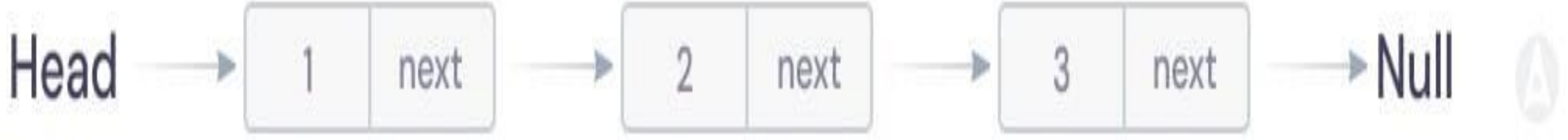
- A queue is a data structure that organizes several pieces of information in the order in which operations are applied to them.
- The order is First in, First out– **FIFO,** which is the same as Last in, Last out **– LILO**.



3. Linked list:

- A linked list (also known as a list) is a linear collection of data elements of any kind, called nodes, with each node having a value and linking to the next in the chain.

- There are different linked lists, such as a singly-linked list, circular linked list, and doubly-linked list.

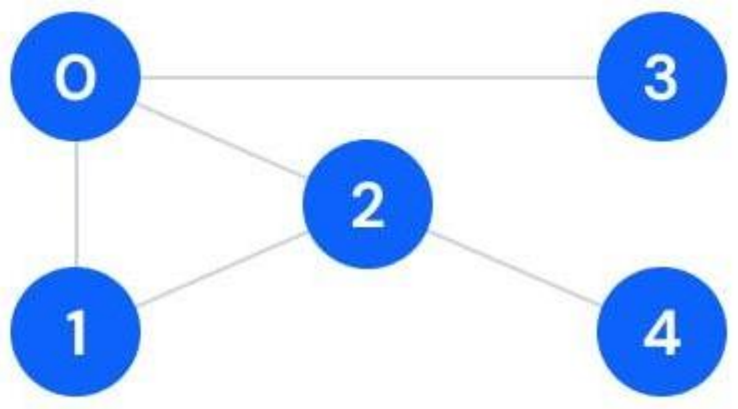


The nonlinear data structures:

1. Graph:

- A graph is a data structure representing relationships between data pieces.

- Graphs often represent social networks and roadmaps.

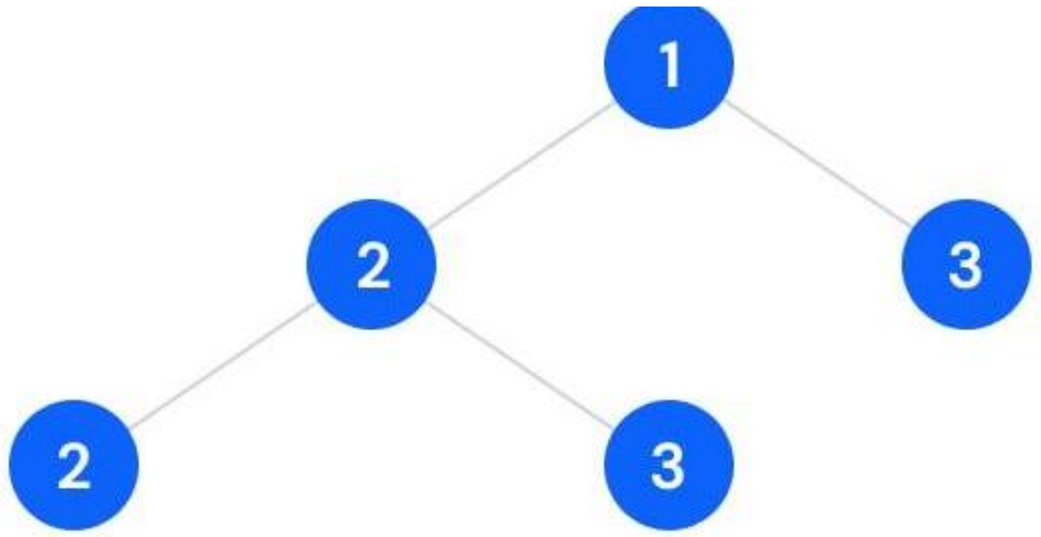


2. **Tree:**

- A tree data structure stores a **Hierarchy of Information**.

- Trees are commonly used to **represent the structure of XML documents and file systems.**

- Many different tree types exist, including binary, B-trees, and AVL trees.



- **Heap:**
  1. A heap is a data structure that stores a sequence of elements.
  2. Heaps implement priority queues.

- **Hash table:**
  1. A hash table is a data structure that stores a mapping of keys to values.
  2. Hash tables are commonly used to implement associative arrays and lookup tables.

Common use cases for data structures:

- **Managing resources and services:** Data structures can be used to keep track of resources and services in a computer system. For example, hash tables can use the IP addresses of all the computers on a network to store data.
- **Maintaining information about users:** Data structures can store information about a software application's users. For example, you can use a linked list to store the names and contact information of all the users of a social networking site.
- **Storing data from sensors:** Data structures can store data like temperature sensors or motion detectors. For example, you can use an array to store the temperatures recorded by a thermometer over time.
- **Indexing:** Indexing is a way of accessing data. Indexing is often used to speed up searches. For example, you can use a binary search tree to index the words in a dictionary so that you can quickly find the definition of a word.
- **Searching:** Data structures can be used to search for data. For example, a binary search tree can quickly find a person's contact information in an extensive database.

Data structure operations

- **Traversal**
  Traversal operations are used to visit each node in a data structure in a specific order. This technique is typically employed for printing, searching, displaying, and reading the data stored in a data structure.
- **Insertion**
  Insertion operations add new data elements to a data structure. You can do this at the data structure's beginning, middle, or end.
- **Deletion**
  Deletion operations remove data elements from a data structure. These operations are typically performed on nodes that are no longer needed.
- **Search**
  Search operations are used to find a specific data element in a data structure. These operations typically employ a compare function to determine if two data elements are equal.
- **Sort**
  Sort operations are used to arrange the data elements in a data structure in a specific order. This can be done using various sorting algorithms, such as insertion sort, bubble sort, merge sort, and quick sort.
- **Merge**
  Merge operations are used to combine two data structures into one. This operation is typically used when two data structures need to be combined into a single structure.
- **Copy**
  Copy operations are used to create a duplicate of a data structure. This can be done by copying each element in the original data structure to the new one.

How to choose a data structure

- **The operations that will be performed:** The choice of data structure should be based on the operations performed. For example, you should use a linked list if you need to perform insertions and deletions. If you need to perform indexing, then you should use an array.
- **The time complexity of the operations:** The choice of data structure should be based on the time complexity of the operations that will be performed. For example, if you need to perform searches frequently, you should use a binary search tree.
- **The space complexity of the operations:** The choice of data structure should be based on the space complexity of the operations that will be performed. For example, if you need to store a lot of data, you should use an array.
- **Memory usage:** The choice of data structure should be based on the amount of memory used. For example, if you need to store a lot of data in memory, you should use a linked list.