DATA STRUCTURES

UNIT-I

Syllabus: Introduction to Linear Data Structures: Definition and importance of linear data structures, Abstract data types (ADTs) and their implementation, Overview of time and space complexity analysis for linear data structures. **Searching Techniques**: Linear & Binary Search, **Sorting Techniques**: Bubble sort, Selection sort, Insertion Sort.

Introduction to Data Structures

- **What is Data?**
- **Data** can be defined as a representation of facts, concepts, or instructions in a formalized manner.
- **Characteristics of Data**

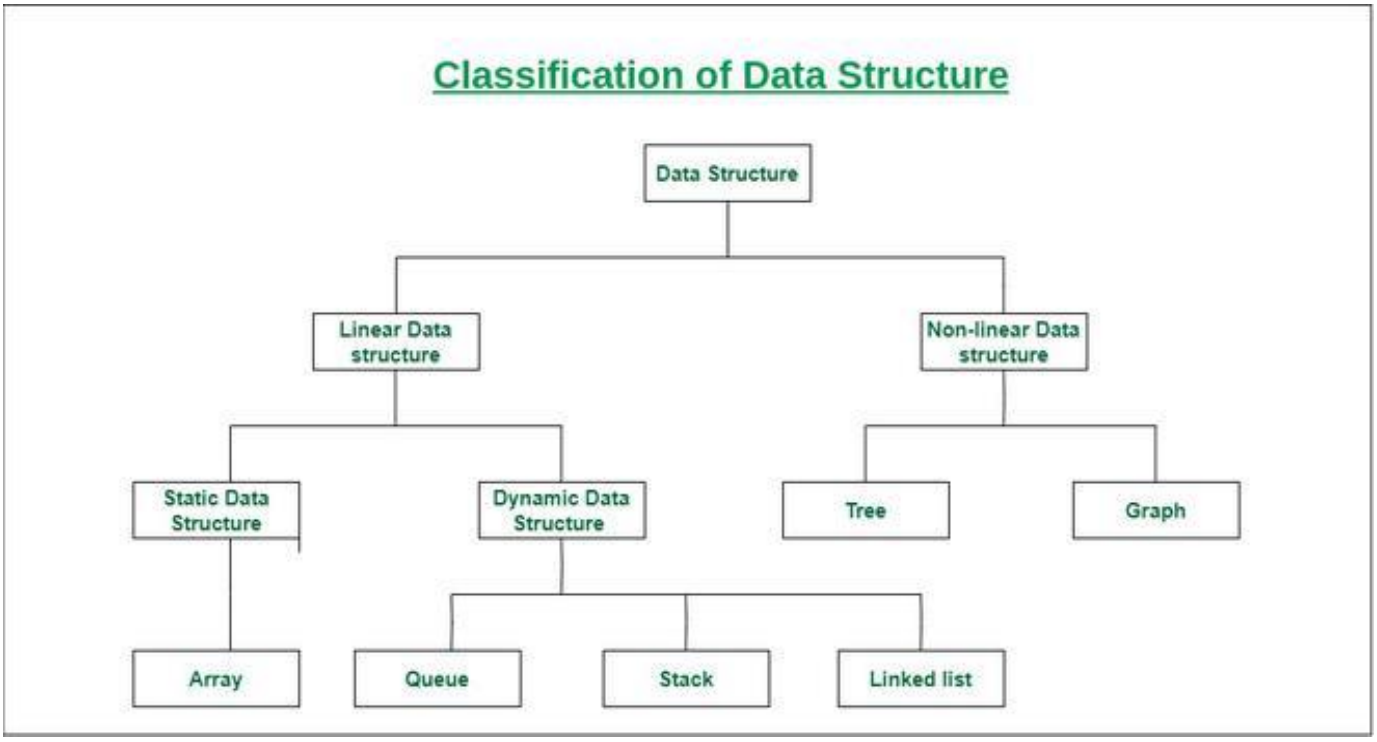
S.No.	Characteristic	What we concern about this Characteristic?
1	Accuracy	Is the information correct in every detail?
2	Completeness	How comprehensive is the information?
3	Reliability	Does the information contradict other trusted resources?
4	Relevance	Do you really need this information?
5	Timeliness	How up- to-date is information? Can it be used for real-time reporting?

- **Examples of Data Volumes:**

Unit	Value	Example
Kilobytes (KB)	1,000 bytes	a paragraph of a text document
Megabytes (MB)	1,000 Kilobytes	a small novel
Gigabytes (GB)	1,000 Megabytes	Beethoven's 5th Symphony
Terabytes (TB)	1,000 Gigabytes	all the X-rays in a large hospital
Petabytes (PB)	1,000 Terabytes	half the contents of all US academic research libraries
Exabytes (EB)	1,000 Petabytes	about one fifth of the words people have ever spoken
Zettabytes (ZB)	1,000 Exabytes	as much information as there are grains of sand on all the world's beaches
Yottabytes (YB)	1,000 Zettabytes	as much information as there are atoms in 7,000 human bodies

- **What is a Data structure?**
- A data structure is a way of organizing and storing data in a computer so that it can be accessed and used efficiently.
- It refers to the logical or mathematical representation of data, as well as the implementation in a computer program.

- **Classification:**
- Data structures can be classified into two broad categories:
- **Linear Data Structure**: A 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.
- Elements are arranged in one dimension, also known as linear dimension.
- Examples are array, stack, queue, etc.
- **Non-linear Data Structure**: Data structures where data elements are not placed sequentially or linearly are called non-linear data structures.
- Elements are arranged in one-many, many-one and many-many dimensions.
- Examples are trees and graphs.

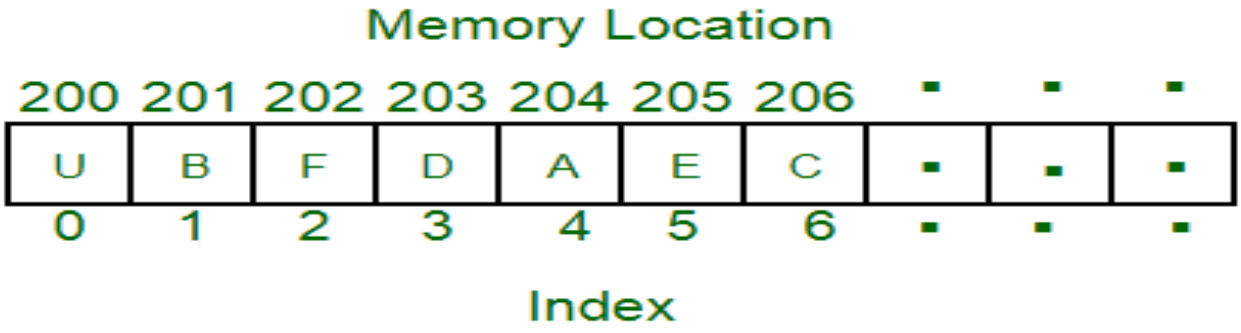


- **Applications of Data Structures:**Data structures are used in a wide range of computer programs and applications, including:
- **Databases:** Data structures are used to organize and store data in a database, allowing for efficient retrieval and manipulation.
- **Operating systems:** Data structures are used in the design and implementation of operating systems to manage system resources, such as memory and files.
- **Computer graphics:** Data structures are used to represent geometric shapes and other graphical elements in computer graphics applications.
- **Artificial intelligence:** Data structures are used to represent knowledge and information in artificial intelligence systems.

- **Advantages of Data Structures:**The use of data structures provides several advantages, including:
- **Efficiency:** Data structures allow for efficient storage and retrieval of data, which is important in applications where performance is critical.
- **Flexibility:** Data structures provide a flexible way to organize and store data, allowing for easy modification and manipulation.
- **Reusability:** Data structures can be used in multiple programs and applications, reducing the need for redundant code**.**
- **Maintainability:** Well-designed data structures can make programs easier to understand, modify, and maintain over time.
- **Need Of Data structure**: Data structures provide an easy way of organizing, retrieving, managing, and storing data.
- **Here is a list of the needs for data structure:**
  - Data structure modification is easy.
  - It requires less time.
  - Save storage memory space.
  - Data representation is easy.
  - Easy access to the large database.

Data Type	Data Structure
➢ The data type is the form of a variable to which a value can be assigned. It defines that the particular variable will assign the values of the given data type only.	➢ Data structure is a collection of different kinds of data. That entire data can be represented using an object and can be used throughout the program.
➢ It can hold value but not data. Therefore, it is dataless.	➢ It can hold multiple types of data within a single object.
➢ The implementation of a data type is known as abstract implementation.	➢ Data structure implementation is known as concrete implementation.
➢ There is no time complexity in the case of data types.	➢ In data structure objects, time complexity plays an important role.
➢ In the case of data types, the value of data is not stored because it only represents the type of data that can be stored.	➢ While in the case of data structures, the data and its value acquire the space in the computer's main memory. Also, a data structure can hold different kinds and types of data within one single object.
➢ Data type examples are int, float, double, etc.	➢ Data structure examples are stack, queue, tree, etc.

- **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.



- **Characteristics of an Array:**An array has various characteristics which are as follows:
- Arrays use **an index-based data structure** which helps to identify each of the elements in an array easily using the index.
- If a user wants to **store multiple values of the same data type**, then the array can be utilized efficiently.
- An array can also **handle complex data structures** by storing data in a **two-dimensional array**.
- An array is also **used to implement other data structures** like Stacks, Queues, Heaps, Hash tables, etc.
- The **search process** in an array can be done **very easily.**
- **Operations performed on array:**
- **Initialization:** An array can be initialized with values at the time of declaration or later using an assignment statement.
- **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.
- **Searching for elements:** Arrays can be searched for a specific element using linear search or binary search algorithms.
- **Sorting elements:** Elements in an array can be sorted in ascending or descending order using algorithms like bubble sort, insertion sort, or quick sort.
- **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.
- **Deleting elements:** Elements can be deleted from an array by shifting the elements that come after it to fill the gap.
- **Updating elements:** Elements in an array can be updated or modified by assigning a new value to a specific index.
- **Traversing elements:** The elements in an array can be traversed in order, visiting each element once.
- **Note:**These are some of the most common operations performed on arrays. The specific operations and algorithms used may vary based on the requirements of the problem and the programming language used.
- **Applications of Array:**Different applications of an array are as follows:
- An array is used in solving matrix problems.
- Database records are also implemented by an array.
- It helps in implementing a sorting algorithm.
- It is also used to implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.
- An array can be used for CPU scheduling.
- Can be applied as a lookup table in computers.

- Arrays can be used in speech processing where every speech signal is an array.
- The screen of the computer is also displayed by an array. Here we use a multidimensional array.
- The array is used in many management systems like a library, students, parliament, etc.
- The array is used in the online ticket booking system. Contacts on a cell phone are displayed by this array.
- In games like online chess, where the player can store his past moves as well as current moves. It indicates a hint of position.
- To save images in a specific dimension in the android Like 360*1200.
- **Real-Life Applications of Array:**
- An array is frequently used to store data for mathematical computations.
- It is used in image processing.
- It is also used in record management.
- Book pages are also real-life examples of an array.
- It is used in ordering boxes as well.

LINEAR DATA STRCUTURES

- *Linear Data Structures are a type of data structure in computer science where data elements are arranged sequentially or linearly.*
- *Each element has previous and next adjacent, except for the first and last elements.*

Characteristics of Linear Data Structure:

- **Sequential Organization:** In linear data structures, data elements are arranged sequentially, one after the other. Each element has a unique predecessor (except for the first element) and a unique successor (except for the last element)
- **Order Preservation:** The order in which elements are added to the data structure is preserved. This means that the first element added will be the first one to be accessed or removed, and the last element added will be the last one to be accessed or removed.
- **Fixed or Dynamic Size**: Linear data structures can have either fixed or dynamic sizes. Arrays typically have a fixed size when they are created, while other structures like linked lists, stacks, and queues can dynamically grow or shrink as elements are added or removed.
- **Efficient Access:** Accessing elements within a linear data structure is typically efficient. For example, arrays offer constant-time access to elements using their index.

Linear data structures are commonly used for organizing and manipulating data in a sequential fashion. Some of the most common linear data structures include:

- **Arrays:** A collection of elements stored in contiguous memory locations.
- **Linked Lists:** A collection of nodes, each containing an element and a reference to the next node.
- **Stacks:** A collection of elements with **Last-In-First-Out (LIFO)** order.
- **Queues:** A collection of elements with **First-In-First-Out (FIFO)** order.

Some of the advantages of linear data structures are:

- **Efficient data access**: Elements can be easily accessed by their position in the sequence. For example, arrays offer constant-time access to elements using their index.
- **Dynamic sizing**: Linear data structures can dynamically adjust their size as elements are added or removed. For example, linked lists, stacks, and queues can grow or shrink as needed.

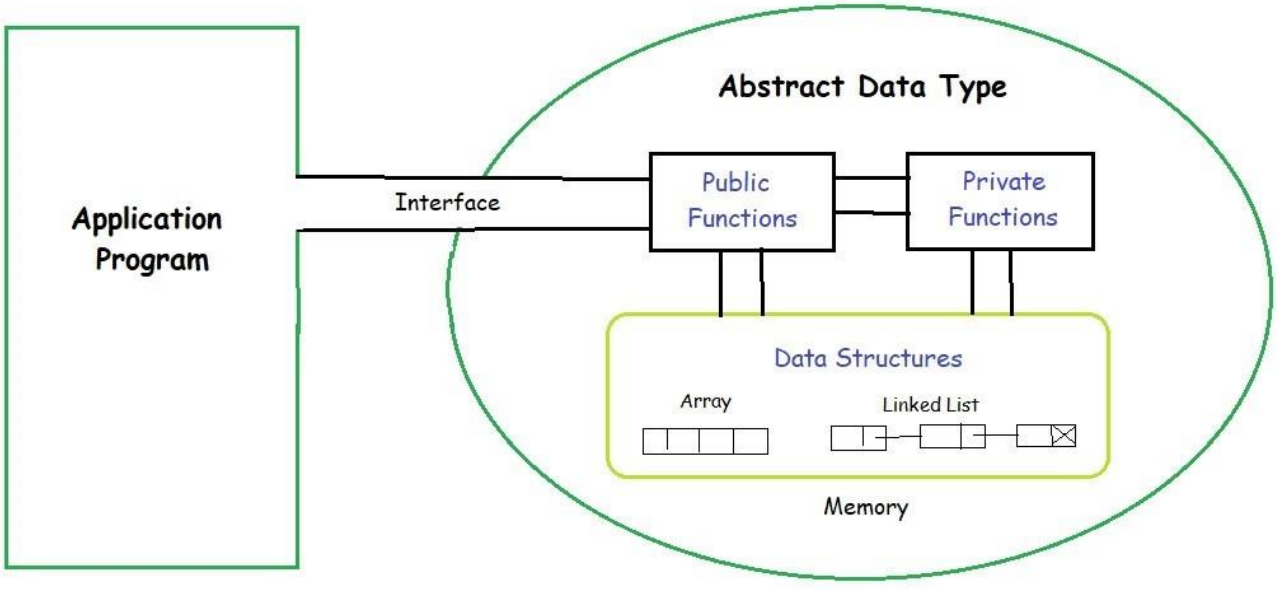
- **Ease of implementation**: Linear data structures can be easily implemented using arrays or linked lists. They are also simpler to understand and manipulate than non-linear data structures.

Disadvantages of linear data structures include[12]:

- Fixed-size: Arrays have a fixed size, which makes them less flexible than other linear data structures.
- Limited functionality: Stacks and queues have limited functionality and can only be used for specific applications.
- Slow insertion and deletion: Linked lists can be slow to insert or delete elements due to the need to traverse the list.
- Poor utilization of computer memory: With the increase in the size of the data structure, the time complexity of the structure increases.

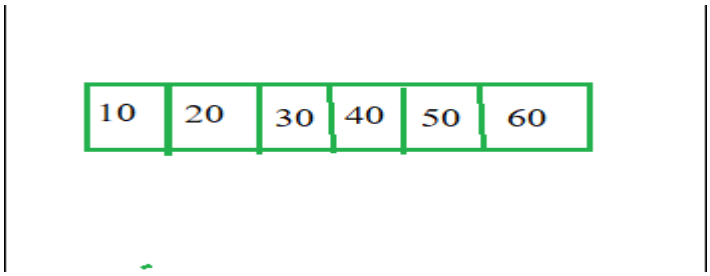
Abstract data types (ADTs) and their implementation:

- Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of values and a set of operations.
- The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.
- It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.
- It is called “abstract” because it gives an implementation-independent view.
- The process of providing only the essentials and hiding the details is known as **abstraction.**



- So a user only needs to know what a data type can do, but not how it will be implemented.
- Think of ADT as a black box which hides the inner structure and design of the data type.
- Now we'll define three ADTs namely List ADT, Stack ADT, Queue ADT.

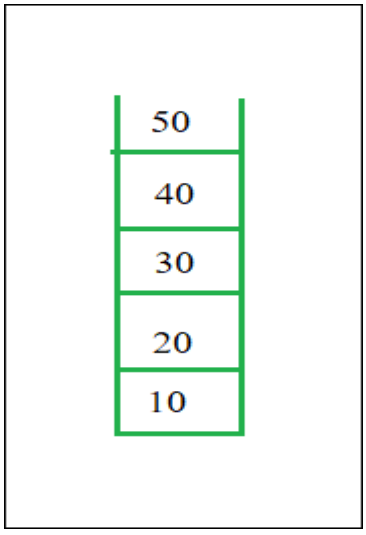
1. List ADT



Vies of list

- The data is generally stored in key sequence in a list which has a head structure consisting of *count*, *pointers* and *address of compare function* needed to compare the data in the list.
- The data node contains the *pointer* to a data structure and a *self-referential pointer* which points to the next node in the list.
- The **List ADT Functions** is given below:
- get() – Return an element from the list at any given position.
- insert() – Insert an element at any position of the list.
- remove() – Remove the first occurrence of any element from a non-empty list.
- removeAt() – Remove the element at a specified location from a non-empty list.
- replace() – Replace an element at any position by another element.
- size() – Return the number of elements in the list.
- isEmpty() – Return true if the list is empty, otherwise return false.
- isFull() – Return true if the list is full, otherwise return false.

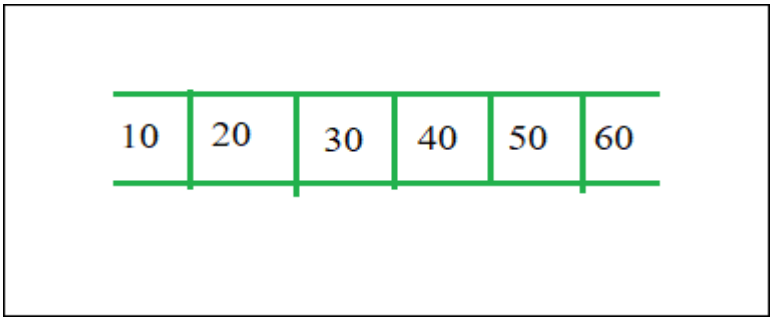
2. Stack ADT



View of stack

- In Stack ADT Implementation instead of data being stored in each node, the pointer to data is stored.
- The program allocates memory for the *data* and *address* is passed to the stack ADT.
- The head node and the data nodes are encapsulated in the ADT. The calling function can only see the pointer to the stack.
- The stack head structure also contains a pointer to *top* and *count* of number of entries currently in stack.
- push() – Insert an element at one end of the stack called top.
- pop() – Remove and return the element at the top of the stack, if it is not empty.
- peek() – Return the element at the top of the stack without removing it, if the stack is not empty.
- size() – Return the number of elements in the stack.
- isEmpty() – Return true if the stack is empty, otherwise return false.
- isFull() – Return true if the stack is full, otherwise return false.

3. Queue ADT



View of Queue

- The queue abstract data type (ADT) follows the basic design of the stack abstract data type.
- Each node contains a void pointer to the *data* and the *link pointer* to the next element in the queue. The program's responsibility is to allocate memory for storing the data.
- enqueue() – Insert an element at the end of the queue.
- dequeue() – Remove and return the first element of the queue, if the queue is not empty.
- peek() – Return the element of the queue without removing it, if the queue is not empty.
- size() – Return the number of elements in the queue.
- isEmpty() – Return true if the queue is empty, otherwise return false.
- isFull() – Return true if the queue is full, otherwise return false.

Features of ADT:

Abstract data types (ADTs) are a way of encapsulating data and operations on that data into a single unit. Some of the key features of ADTs include:

- **Abstraction:** The user does not need to know the implementation of the data structure only essentials are provided.

- **Better Conceptualization:** ADT gives us a better conceptualization of the real world.
- **Robust:** The program is robust and has the ability to catch errors.
- **Encapsulation**: ADTs hide the internal details of the data and provide a public interface for users to interact with the data. This allows for easier maintenance and modification of the data structure.
- **Data Abstraction**: ADTs provide a level of abstraction from the implementation details of the data. Users only need to know the operations that can be performed on the data, not how those operations are implemented.
- **Data Structure Independence**: ADTs can be implemented using different data structures, such as arrays or linked lists, without affecting the functionality of the ADT.
- **Information Hiding:** ADTs can protect the integrity of the data by allowing access only to authorized users and operations. This helps prevent errors and misuse of the data.
- **Modularity**: ADTs can be combined with other ADTs to form larger, more complex data structures. This allows for greater flexibility and modularity in programming.

Overall, ADTs provide a powerful tool for organizing and manipulating data in a structured and efficient manner.

Abstract data types (ADTs) have several advantages and disadvantages that should be considered when deciding to use them in software development. Here are some of the main advantages and disadvantages of using ADTs:

Advantages:

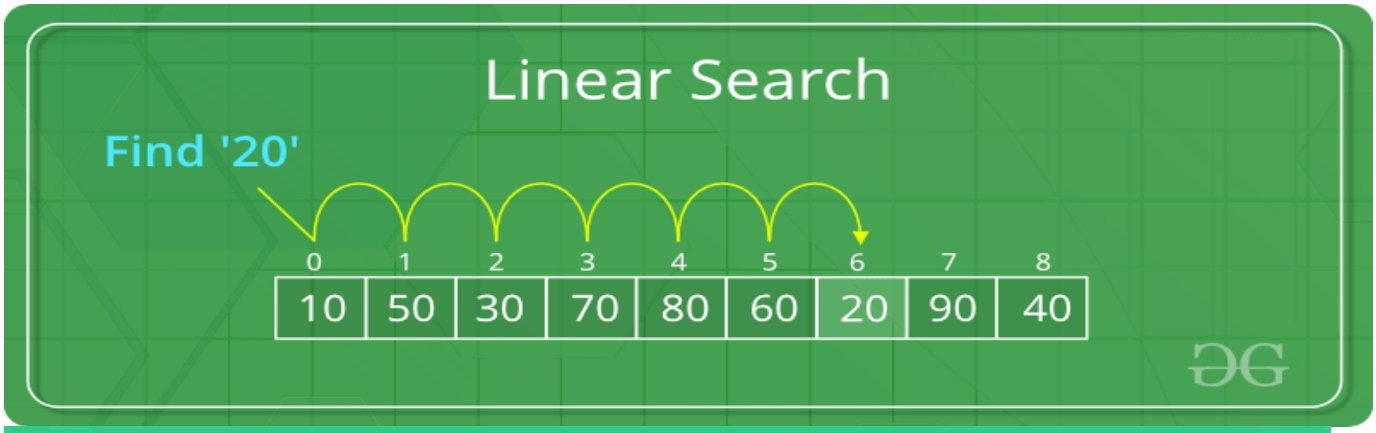
- **Encapsulation**: ADTs provide a way to encapsulate data and operations into a single unit, making it easier to manage and modify the data structure.
- **Abstraction**: ADTs allow users to work with data structures without having to know the implementation details, which can simplify programming and reduce errors.
- **Data Structure Independence**: ADTs can be implemented using different data structures, which can make it easier to adapt to changing needs and requirements.
- **Information Hiding**: ADTs can protect the integrity of data by controlling access and preventing unauthorized modifications.
- **Modularity**: ADTs can be combined with other ADTs to form more complex data structures, which can increase flexibility and modularity in programming.

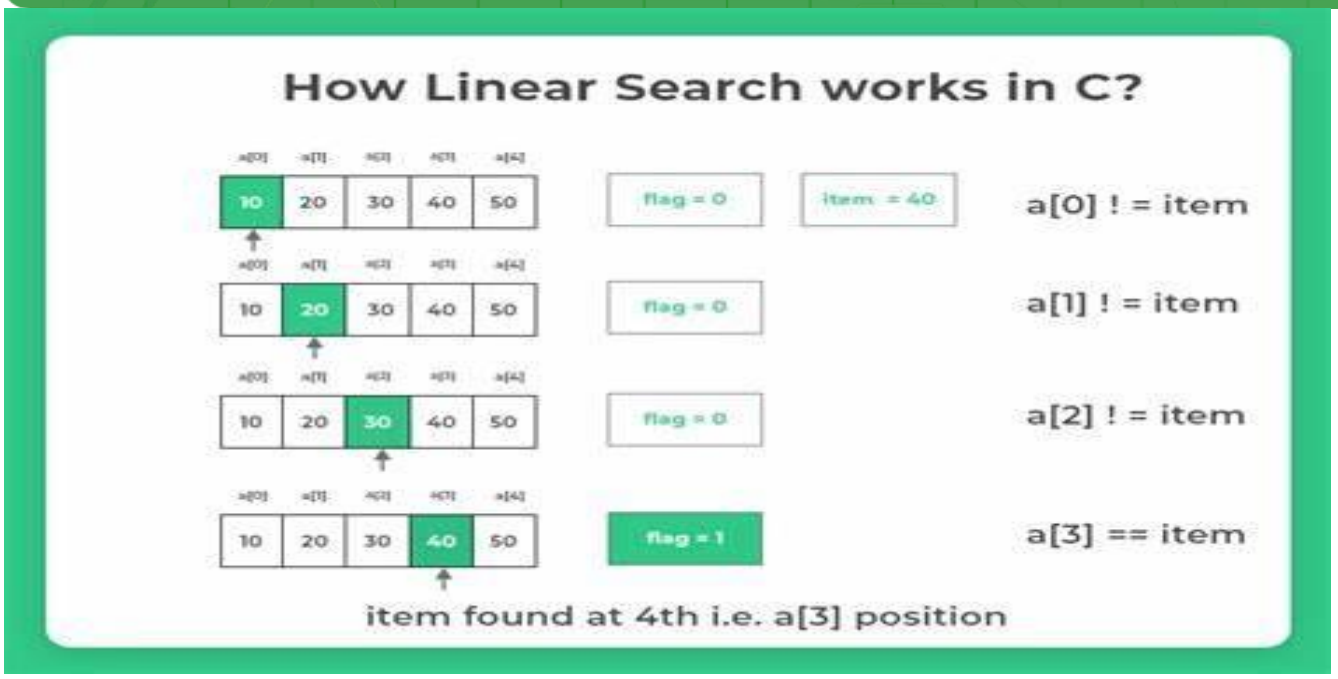
Disadvantages:

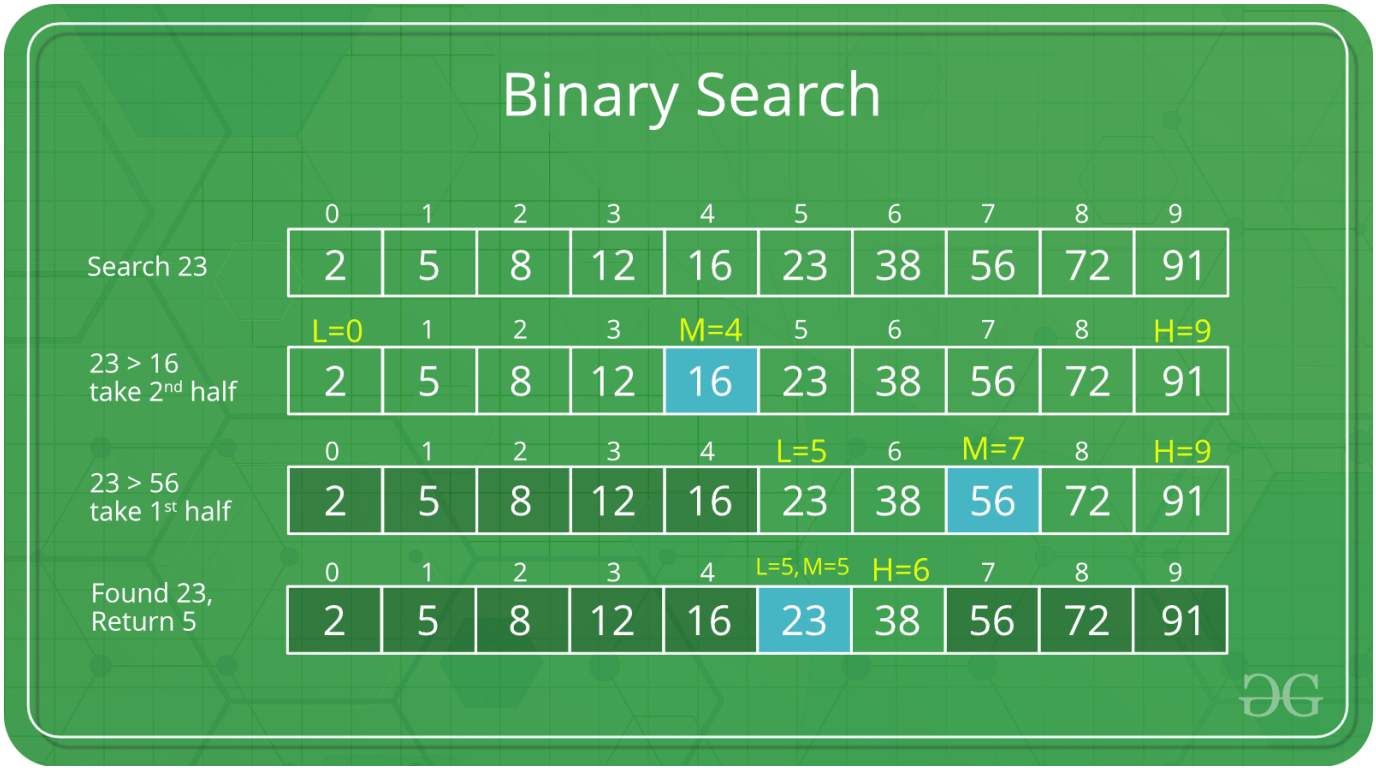
- **Overhead**: Implementing ADTs can add overhead in terms of memory and processing, which can affect performance.
- **Complexity**: ADTs can be complex to implement, especially for large and complex data structures.
- **Learning** Curve: Using ADTs requires knowledge of their implementation and usage, which can take time and effort to learn.
- **Limited Flexibility:** Some ADTs may be limited in their functionality or may not be suitable for all types of data structures.
- **Cost**: Implementing ADTs may require additional resources and investment, which can increase the cost of development.

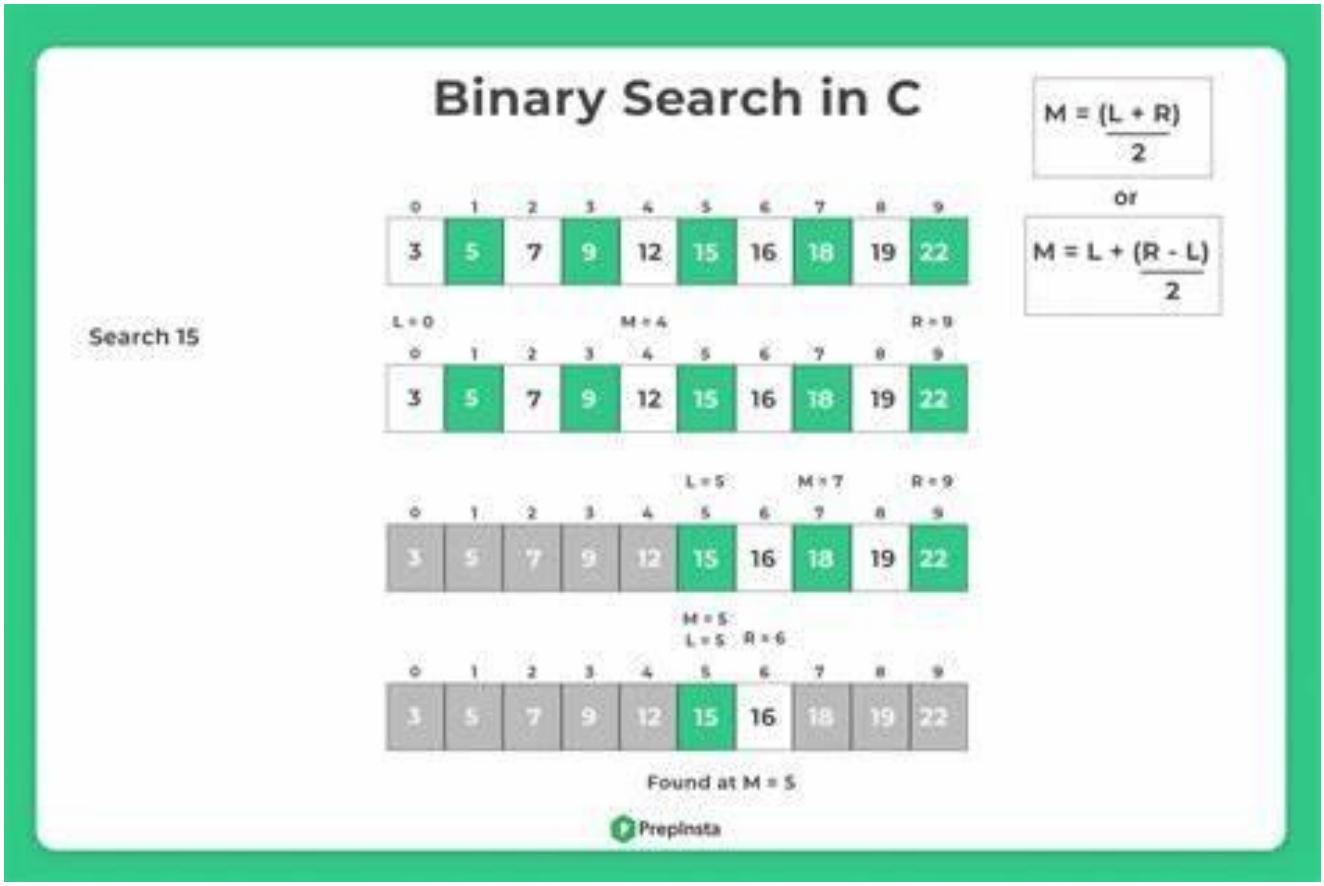
Overview of time and space complexity analysis for linear data structures:

	Access	Search	Insertion	Deletion
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))
Splay Tree	-	Θ(log(n))	Θ(log(n))	Θ(log(n))
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))

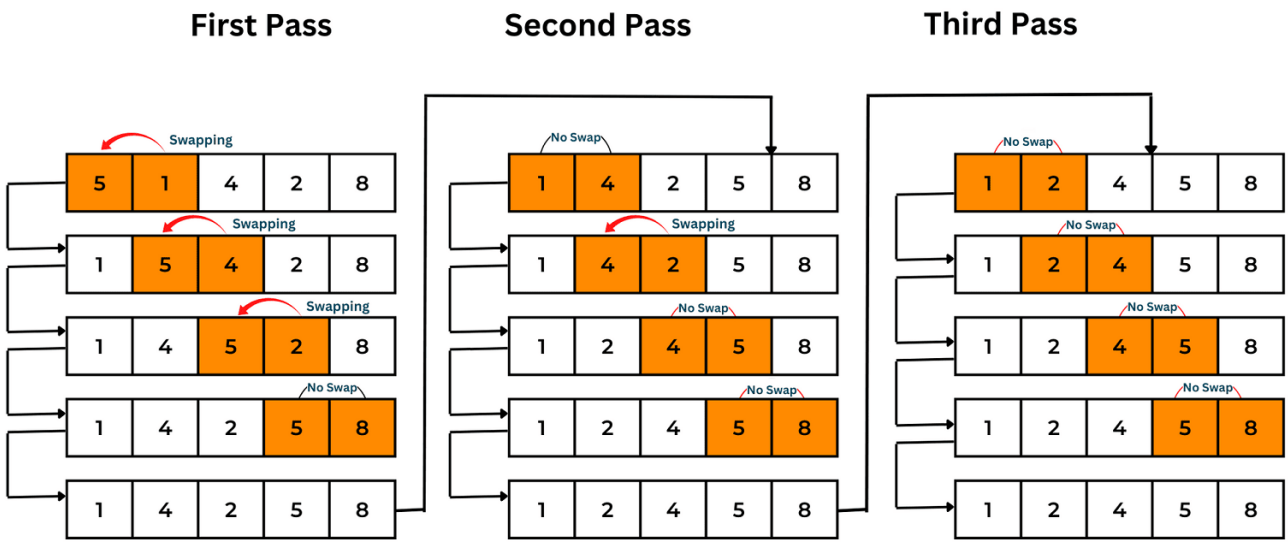


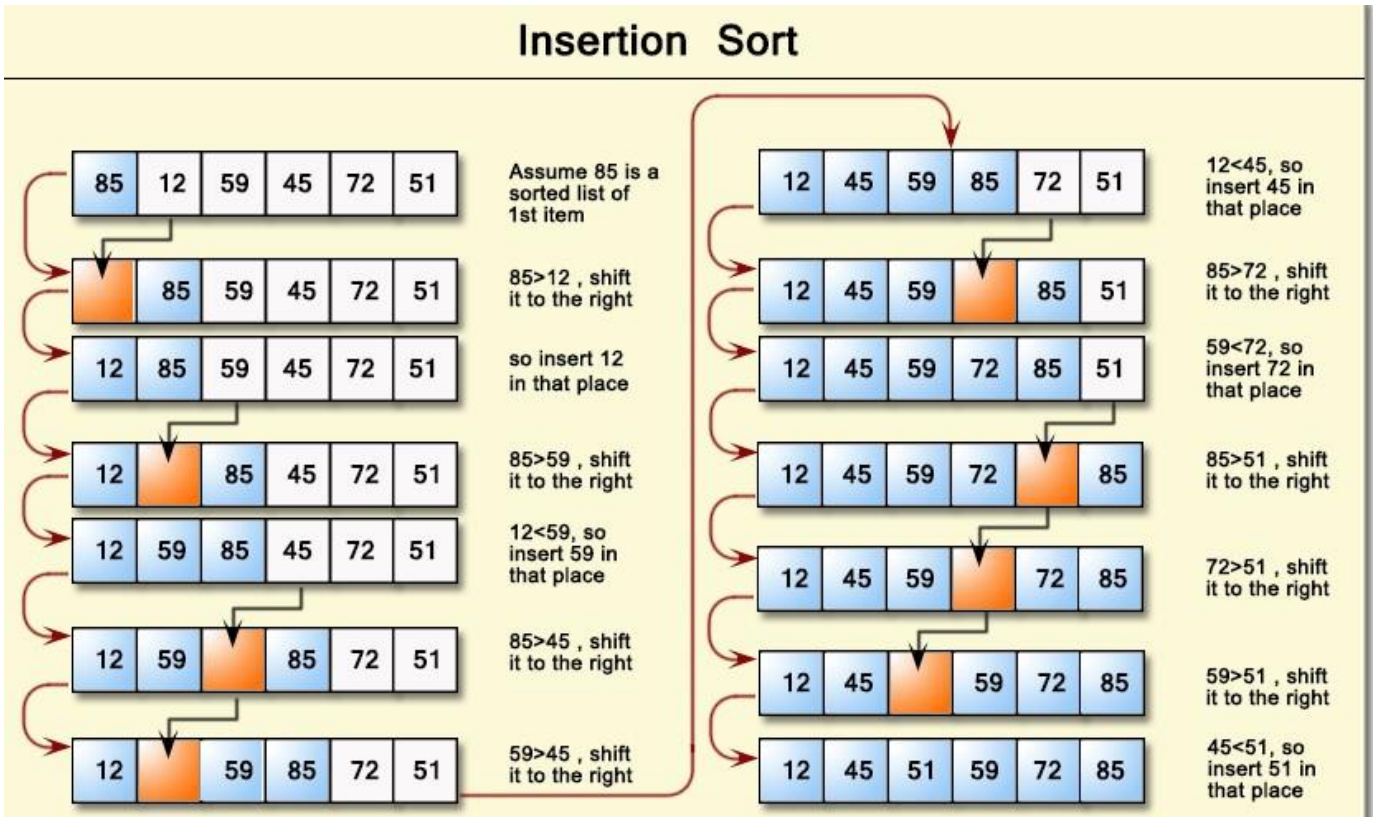



Binary Search
Search 23
23 > 16
take 2nd half
23 > 56
take 1st half
Found 23,
Return 5
0 1 2 3 4 5 6 7 8 9
2 5 8 12 16 23 38 56 72 91
L=0 M=4 H=9
L=5 M=7 H=9
L=5, M=5 H=6

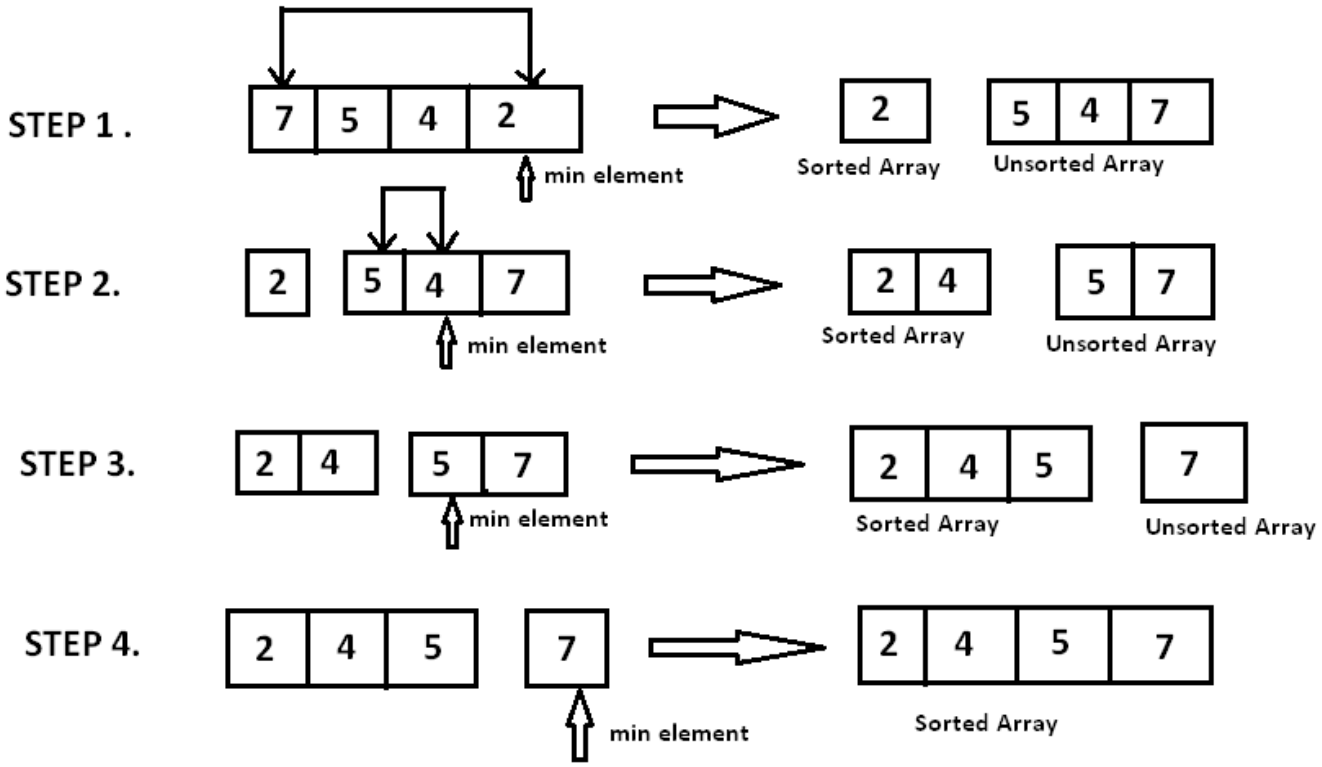

Binary Search in C
M = (L + R) / 2
or
M = L + (R - L) / 2
Search 15
3 5 7 9 12 15 16 18 19 22
L = 0 M = 4 R = 9
L = 5 M = 7 R = 9
M = 5 L = 5 R = 6
Found at M = 5
Prepinsta

BUBBLE SORTING





Selection sort:



Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity
Linear Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)
Bubble Sort	O(n)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Insertion Sort	O(n)	O(n^2)	O(n^2)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)