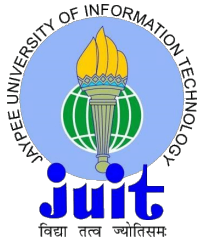


DATA STRUCTURES

(Course Code: 25B11CI318)

Overview

Mr. Sandeep Kumar Patel

Assistant Professor

Jaypee University of Information Technology

Waknaghat, Solan, HP-173234

What is a Data Structure?

Definition

A **data structure** is a logical or mathematical model that defines how data is organized, stored, and related, along with the operations that can be performed on it to achieve efficient computation.

Formal Definition (Horowitz & Sahni)

A data structure is a collection of data elements whose organization is characterized by the relationships among the data elements and the operations that can be performed on the data.

Mathematical Representation

A data structure d is represented as

$$\boxed{d = (D, F, A)}$$

where

D – Domain / Data Values

A finite set of data elements (domain of values).

A – Axiom/ Rules

A set of rules that specify how the functions operate and interact with the data values.

F – Functions / Operations

A set of operations that manipulate the data.

- Insert, Delete,
- Search,
- Traverse
- Update

Example: Linked List

Consider the linked list

$$10 \longrightarrow 20 \longrightarrow 30 \longrightarrow \texttt{NULL}$$

Example: Linked List

Consider the linked list

$$10 \longrightarrow 20 \longrightarrow 30 \longrightarrow \texttt{NULL}$$

Data (D):

Example: Linked List

Consider the linked list

$$10 \longrightarrow 20 \longrightarrow 30 \longrightarrow \texttt{NULL}$$

Data (D): $\{10, 20, 30\}$

Operations (F):

Example: Linked List

Consider the linked list

$$10 \longrightarrow 20 \longrightarrow 30 \longrightarrow \texttt{NULL}$$

Data (D): $\{10, 20, 30\}$

Operations (F): $\{\text{Insert}, \text{Delete}, \text{Search}, \text{Traverse}, \text{Update}\}$

Axiom (A):

Example: Linked List

Consider the linked list

$$10 \longrightarrow 20 \longrightarrow 30 \longrightarrow \texttt{NULL}$$

Data (D): $\{10, 20, 30\}$

Operations (F): $\{\text{Insert}, \text{Delete}, \text{Search}, \text{Traverse}, \text{Update}\}$

Axiom (A): Rule: To get to 30, you must pass through 10 and 20 first.

$$\boxed{\mathit{LinkedList} = (D, F, A)}$$

Example: Stack

Consider the following stack:

Example: Stack

Consider the following stack:

Data (D): $\{10, 20, 30\}$

Axiom (A): Elements are organized as **LIFO** (Last-In, First-Out) principle.

Functions (F): $\{\text{Push}, \text{Pop}, \text{Peek}, \text{IsEmpty}, \text{IsFull}\}$

$$\boxed{Stack = (D, F, A)}$$

Classification of Data Structures

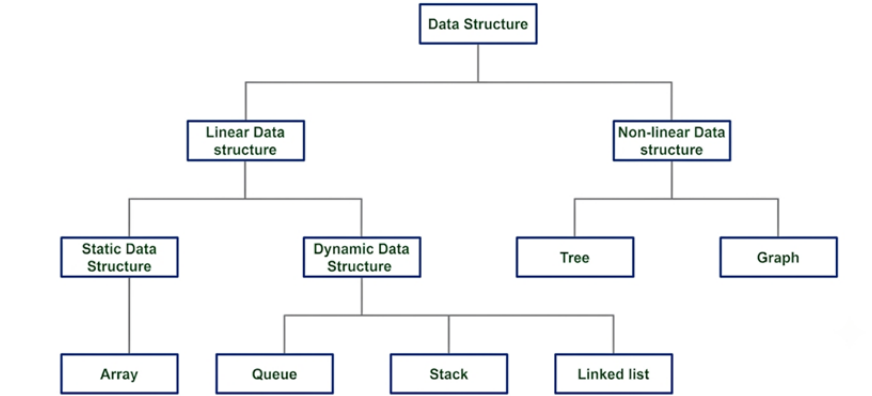


Figure: Classification of Data Structures

Classification of Data Structures

Data structures are broadly classified into two categories:

1. **Linear Data Structures**
   - Elements are arranged sequentially.
   - Each element (except the first and last) has a unique predecessor and successor.
2. **Non-linear Data Structures**
   - Elements are organized hierarchically or as a network.
   - One element can be connected to multiple elements.

Linear Data Structures

A **Linear Data Structure** stores elements in a sequential order.

Types

- **Static Data Structure**
  - Memory size is fixed.
  - Example: **Array**
- **Dynamic Data Structure**
  - Memory size can grow or shrink during execution.
  - Examples:
    - Linked List
    - Stack
    - Queue

Non-linear Data Structures

A **Non-linear Data Structure** organizes data in a hierarchical or network form.

Examples include:

- **Tree**
  - Hierarchical organization.
  - Used in file systems, databases, expression trees.
- **Graph**
  - Collection of vertices connected by edges.
  - Used in social networks, maps, routing algorithms.

What is an Array?

Definition

An array is a collection of elements of the same data type stored in **contiguous memory locations**.

What is an Array?

Definition

An array is a collection of elements of the same data type stored in **contiguous memory locations**.

Example

$$A = \{10, 20, 30, 40, 50\}$$

What is an Array?

Definition

An array is a collection of elements of the same data type stored in **contiguous memory locations**.

Example

$$A = \{10, 20, 30, 40, 50\}$$

Properties

- Same data type
- Fixed size
- Continuous memory
- Indexed access

Array Representation

10	20	30	40	50
0	1	2	3	4

Index

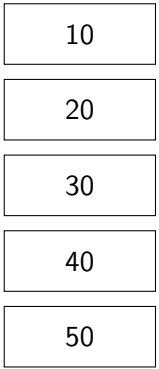
Each element is accessed using its index.

$$A[0] = 10$$

$$A[3] = 40$$

Characteristics of Arrays

- Homogeneous
- Fixed size
- Random access
- Contiguous memory
- Cache friendly



Memory Representation of an Array

Example:

$$int\ A[5] = \{10, 20, 30, 40, 50\}$$

Assume:

- Base Address = 1000
- Size of Integer = 4 Bytes

Index 0	Index 1	Index 2	Index 3	Index 4
10	**20**	**30**	**40**	**50**
1000	1004	1008	1012	1016

Observation

Array elements are stored in **contiguous (adjacent)** memory locations.

Address Calculation Formula

For a one-dimensional array,

$$\boxed{Address(A[i]) = Base + i \times Size}$$

Address Calculation Formula

For a one-dimensional array,

$$\boxed{Address(A[i]) = Base + i \times Size}$$

Example

$$Base = 1000, \qquad Size = 4$$

Find address of

$$A[3]$$

Address Calculation Formula

For a one-dimensional array,

$$\boxed{Address(A[i]) = Base + i \times Size}$$

Example

$$Base = 1000, \qquad Size = 4$$

Find address of

$$A[3]$$

$$Address(A[3]) = 1000 + 3 \times 4 = 1012$$

Address Calculation Formula

For a one-dimensional array,

$$\boxed{Address(A[i]) = Base + i \times Size}$$

Example

$$Base = 1000, \qquad Size = 4$$

Find address of

$$A[3]$$

$$Address(A[3]) = 1000 + 3 \times 4 = 1012$$

Important

Index always starts from 0.

Declaration and Initialization

Declaration

```
1 int A[5];
2 float marks[100];
3 char name[20];
```

Declaration and Initialization

Declaration

```
1 int A[5];
2 float marks[100];
3 char name[20];
```

Initialization

```
1 int A[5]={10,20,30,40,50};
2
3 int B[]={5,10,15,20};
4
5 int C[5]={1,2};
```

Declaration and Initialization

Declaration

```
int A[5];
float marks[100];
char name[20];
```

Initialization

```
int A[5]={10,20,30,40,50};

int B[]={5,10,15,20};

int C[5]={1,2};
```

Note

If fewer values are provided, the remaining elements are initialized to zero.

Traversal of an Array

Traversal means visiting every element exactly once.

```
for(int i=0;i<n;i++)
{
    printf("%d ",A[i]);
}
```

Traversal of an Array

Traversal means visiting every element exactly once.

```
for(int i=0;i<n;i++)
{
    printf("%d ",A[i]);
}
```

Example

$$10 \rightarrow 20 \rightarrow 30 \rightarrow 40 \rightarrow 50$$

Traversal of an Array

Traversal means visiting every element exactly once.

```
for(int i=0;i<n;i++)
{
    printf("%d ",A[i]);
}
```

Example

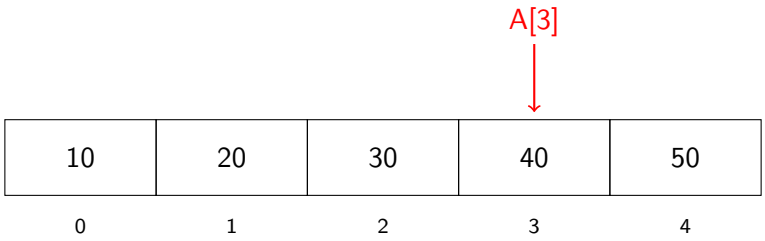
$$10 \rightarrow 20 \rightarrow 30 \rightarrow 40 \rightarrow 50$$

$$\boxed{\textit{Time Complexity} = O(n)}$$

Accessing Array Elements

Given

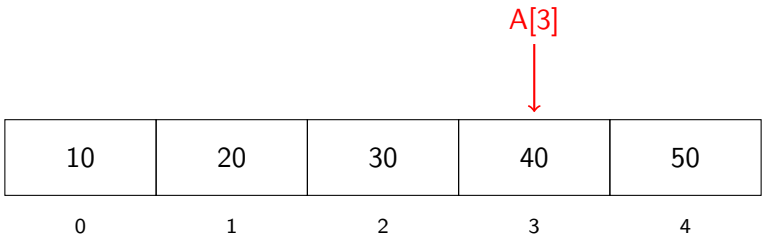
$$A = \{10, 20, 30, 40, 50\}$$



Accessing Array Elements

Given

$$A = \{10, 20, 30, 40, 50\}$$

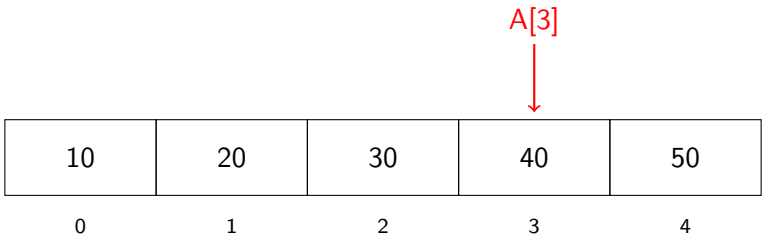


$$A[0] = 10$$

Accessing Array Elements

Given

$$A = \{10, 20, 30, 40, 50\}$$



$$A[0] = 10$$

Updating an Array Element

Suppose

$$A = \{10, 20, 30, 40, 50\}$$

Replace

$$40 \rightarrow 45$$

Updating an Array Element

Suppose

$$A = \{10, 20, 30, 40, 50\}$$

Replace

$$40 \rightarrow 45$$

```
A[3]=45;
```

Updating an Array Element

Suppose

$$A = \{10, 20, 30, 40, 50\}$$

Replace

$$40 \rightarrow 45$$

```
A[3]=45;
```

Updated Array

10 20 30 45 50

Updating an Array Element

Suppose

$$A = \{10, 20, 30, 40, 50\}$$

Replace

$$40 \rightarrow 45$$

```
A[3]=45;
```

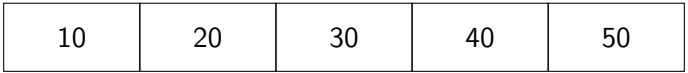
Updated Array

10 20 30 45 50

$$\boxed{\textit{Time Complexity} = O(1)}$$

Insertion in an Array

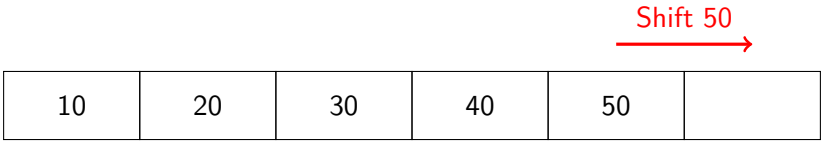
Insert 25 at Position 2



Current Array

Insertion in an Array

Insert 25 at Position 2



Time Complexity $= O(n)$

Insertion in an Array

Insert 25 at Position 2

Insertion Completed

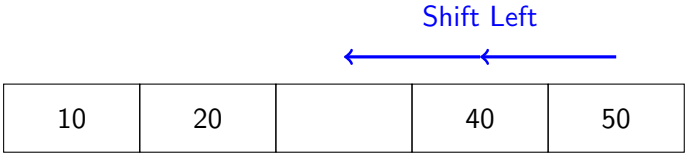
$$\boxed{\text{Time Complexity } = O(n)}$$

Deletion from an Array

Delete 30

Deletion from an Array

Delete 30



$O(n)$

Deletion from an Array

Delete 30

Deletion Completed

$O(n)$

Linear Search

Search Key = 40

Linear Search

Search Key = 40

Algorithm

```
for(i=0;i<n;i++)
{
 if(A[i]==key)
   return i;
}
```

Linear Search

Search Key = 40

Algorithm

```
for(i=0;i<n;i++)
{
 if(A[i]==key)
   return i;
}
```

Time Complexity

$O(n)$

Binary Search

Sorted Array

10	20	30	40	50	60	70

Search = 60

Binary Search

Sorted Array

10 20 30 40 50 60 70

Search = 60

Step 1

Middle =40

Discard Left Half

Binary Search

Sorted Array

10 20 30 40 50 60 70

Search = 60

Step 1

Middle =40

Discard Left Half

Step 2

Remaining

50 60 70

Middle =60; Found

Binary Search

Sorted Array

10 20 30 40 50 60 70

Search = 60

Step 1

Middle =40

Discard Left Half

Step 2

Remaining

50 60 70

Middle =60; Found

$O(\log n)$

Time Complexity Analysis

Operation	Best	Worst
Access	O(1)	O(1)
Traversal	O(n)	O(n)
Linear Search	O(1)	O(n)
Binary Search	O(1)	O(log n)
Insertion	O(1)	O(n)
Deletion	O(1)	O(n)
Updating	O(1)	O(1)

Time Complexity Analysis

Operation	Best	Worst
Access	O(1)	O(1)
Traversal	O(n)	O(n)
Linear Search	O(1)	O(n)
Binary Search	O(1)	O(log n)
Insertion	O(1)	O(n)
Deletion	O(1)	O(n)
Updating	O(1)	O(1)

Insertion and deletion require shifting elements.

Advantages and Disadvantages

Advantages

- Random Access
- Simple Implementation
- Cache Friendly
- Less Memory Overhead
- Fast Retrieval

Disadvantages

- Fixed Size
- Expensive Insertion
- Expensive Deletion
- Memory Wastage
- Overflow Possible

Advantages and Disadvantages

Advantages

- Random Access
- Simple Implementation
- Cache Friendly
- Less Memory Overhead
- Fast Retrieval

Disadvantages

- Fixed Size
- Expensive Insertion
- Expensive Deletion
- Memory Wastage
- Overflow Possible

Question

If insertion and deletion are expensive,
what data structure can solve this problem?

Advantages and Disadvantages

Advantages

- Random Access
- Simple Implementation
- Cache Friendly
- Less Memory Overhead
- Fast Retrieval

Disadvantages

- Fixed Size
- Expensive Insertion
- Expensive Deletion
- Memory Wastage
- Overflow Possible

Question

If insertion and deletion are expensive,
what data structure can solve this problem?

Linked List