DATA STRUCTURES AND ALGORITHM

Module

3

- To identify the difference between array and linked list.
- To determine the importance of linked list
- To enumerate the different types of linked list

LINKED LIST

LINKED LIST

- a collection of *nodes* that together form a linear ordering.

- Each node is divided into two parts:
  a) The information of the element
  b) The address of the next node in the linked list

Address part of the node is called as Linked or next field.

LINKED LIST

Comparison between Array and Linked List

	ARRAY	LINKED LIST
Memory Size	Fixed Size	takes up only as much space in memory as is needed for the length of the list
Memory Location	Stored in successive memory location	Not stored in successive memory location
Allocated Space	Same amount of allocated space	The list expands or contracts as you add or delete elements.

LINKED LIST

- **ARRAY**

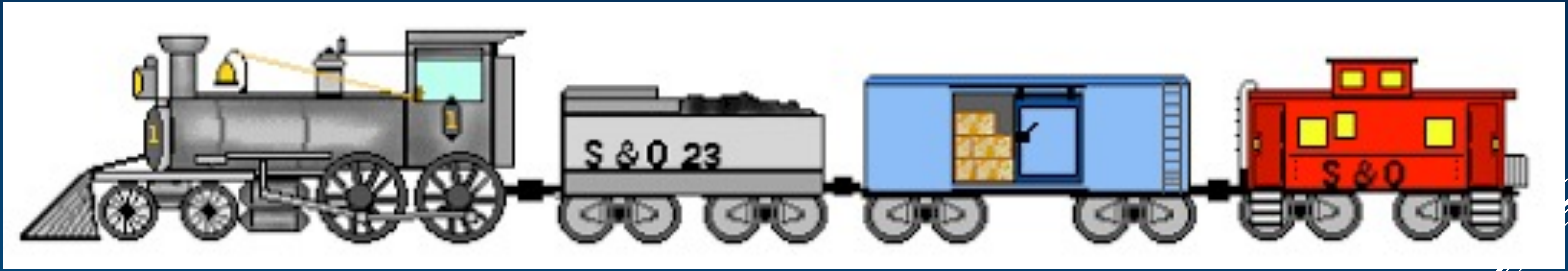
Array elements stored in successive memory locations. Also, order of elements stored in array is same logically and physically

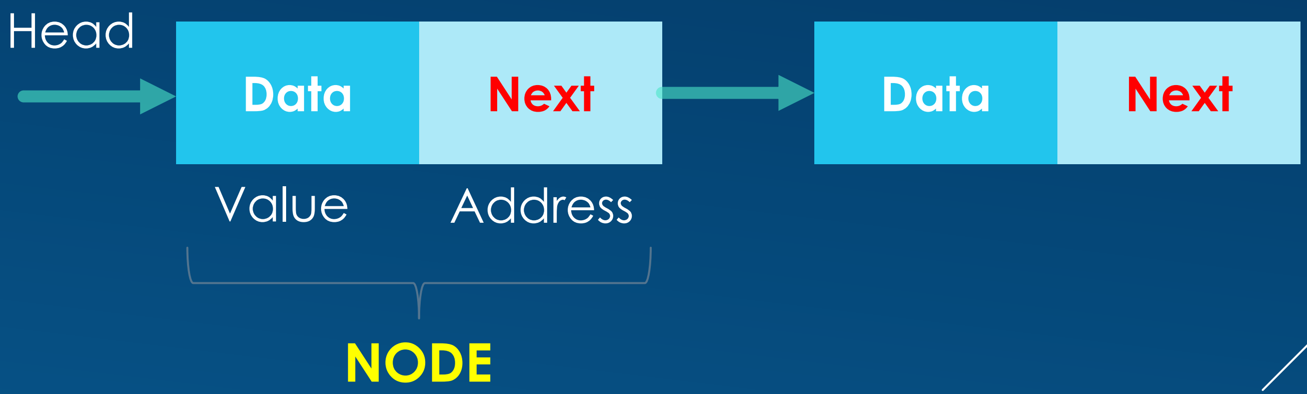
- **LINKED LIST**

Linked List elements can be added to (or deleted from) either end, or added to (or deleted from)the middle of the list.

LINKED LIST

LINKED LIST





LINKED LIST

ADVANTAGES OF A LINKED LIST

1. A dynamic structure
2. Efficient memory utilization
3. Insertion and deletion are easier and efficient
4. Many complex applications can be easily carried out with linked list.

DISADVANTAGES OF A LINKED LIST

1. More memory
2. Access to an arbitrary data item is little bit cumbersome and also time consuming.

LINKED LIST

The primitive operations performed on the linked list are as follows

1. Creation
2. Insertion
3. Deletion
4. Traversing
5. Searching
6. Concatenation

LINKED LIST

The primitive operations performed on the linked list are as follows

1. Creation operation is used to create a linked list. Once a linked list is created with one node, insertion operation can be used to add more elements in a node.

LINKED LIST

The primitive operations performed on the linked list are as follows

2. Insertion operation is used to insert a new node at any specified location in the linked list.

A new node may be inserted.

(*a*) At the beginning of the linked list
(*b*) At the end of the linked list
(c) At any specified position in between in a linked list

LINKED LIST

The primitive operations performed on the linked list are as follows

3. Deletion operation is used to delete an item (or node) from the linked list.

A node may be deleted from the

(*a*) Beginning of the linked list
(*b*) End of the linked list
(c) Specified location of a linked list

LINKED LIST

The primitive operations performed on the linked list are as follows

4. Traversing is the process of going through all the nodes from one end to another end of a linked list.

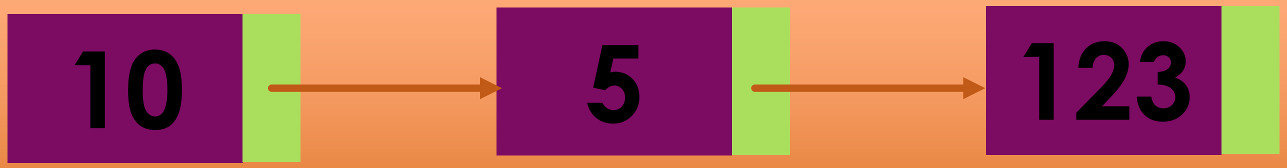
- Singly linked list, can visit from left to right, forward traversing, nodes only.
- Doubly linked list forward and backward traversing is possible.

5. Concatenation is the process of appending the second list to the end of the first list.

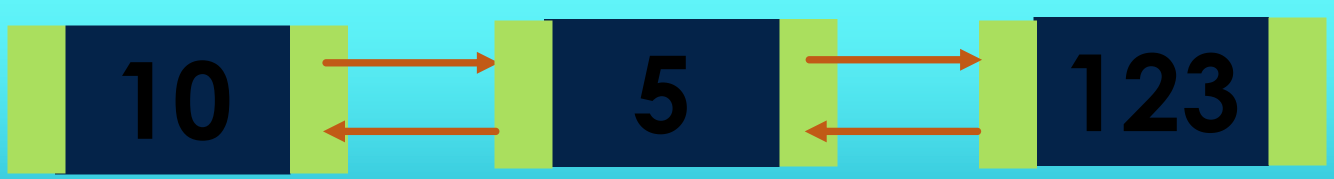
LINKED LIST

TYPES OF LINKED LIST

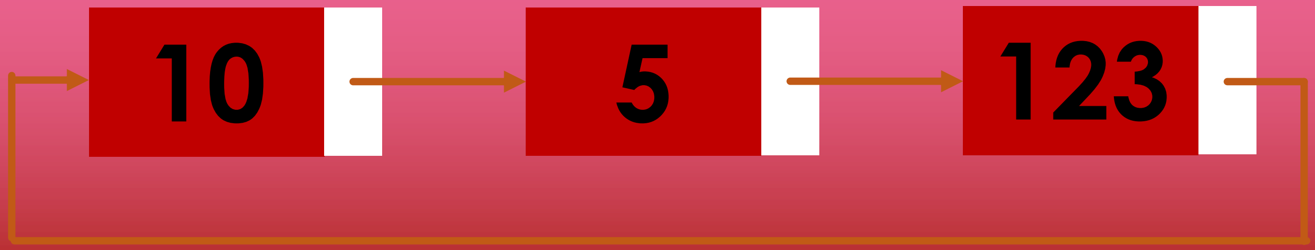
Singly Linked List



Doubly Linked List



Circular Linked List



SINGLY LINKED LIST

All the nodes in a singly linked list are arranged sequentially by linking with a pointer.

ALGORITHM FOR INSERTING OF A NODE AT THE BEGINNING

```
1. Input DATA to be inserted
2. Create a NewNode
3. NewNode → DATA = DATA
4. If (START equal to NULL)
        (a) NewNode → Link = NULL
5. Else
        (a) NewNode → Link = START
6. START = NewNode
7. Exit
```

SINGLY LINKED LIST

ALGORITHM FOR INSERTING OF A NODE AT THE END

```
1. Input DATA to be inserted
2. Create a NewNode
3. NewNode → DATA = DATA
4. NewNode → Next = NULL
5. If (START equal to NULL)
        (a) START = NewNode
6. Else
        (a) TEMP = START
        (b) While (TEMP → Next not equal to NULL)
                TEMP = TEMP → Next
7. TEMP → Next = NewNode
8. Exit
```

SINGLY LINKED LIST

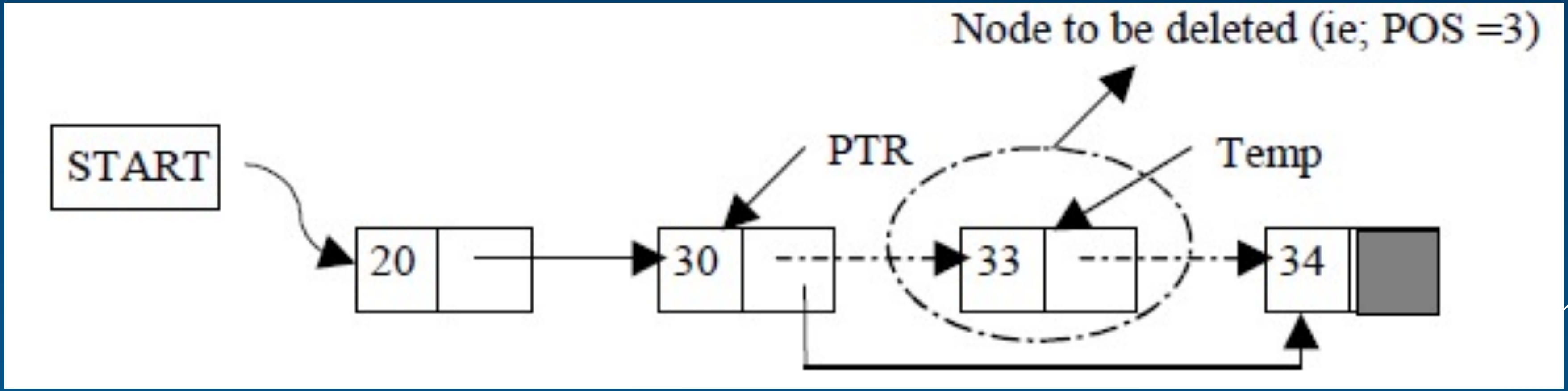
ALGORITHM FOR INSERTING OF A NODE AT ANY SPECIFIED POSITION

```
1. Input DATA and POS (position) to be inserted
2. initialize TEMP = START; and j = 0
3. Repeat the step 3 while( k is less than POS)
        (a) TEMP = TEMP.Next
        (b) If (TEMP is equal to NULL)
                (i) Display “Node in the list less than the position”
                (ii) Exit
        (c) k = k + 1
4. Create a New Node
5. NewNode → DATA = DATA
6. NewNode → Next = TEMP → Next
7. TEMP → Next = NewNode
8. Exit
```

SINGLY LINKED LIST

ALGORITHM FOR DELETING A NODE

Suppose START is the first position in linked list.
Let DATA be the element to be deleted.
TEMP, **HOLD** is a temporary pointer to hold the node address.



SINGLY LINKED LIST

ALGORITHM FOR DELETING A NODE

```
1. Input the DATA to be deleted
2. if ((START $\rightarrow$ DATA) is equal to DATA)
        (a) TEMP = START
        (b) START = START $\rightarrow$ Next
        (c) Set free the node TEMP, which is deleted
        (d) Exit
3. HOLD = START
```

SINGLY LINKED LIST

ALGORITHM FOR DELETING A NODE

```
4. while ((HOLD → Next → Next) not equal to NULL))
        (a) if ((HOLD → NEXT → DATA) equal to DATA)
                (i) TEMP = HOLD → Next
                (ii) HOLD → Next = TEMP → Next
                (iii) Set free the node TEMP, which is
                      deleted
                (iv) Exit
        (b) HOLD = HOLD → Next
```

SINGLY LINKED LIST

ALGORITHM FOR DELETING A NODE

```
5. if ((HOLD → next → DATA) == DATA)
        (a) TEMP = HOLD → Next
        (b) Set free the node TEMP, which is deleted
        (c) HOLD → Next = NULL
        (d) Exit
6. Display "DATA not found"
7. Exit
```

SINGLY LINKED LIST

ALGORITHM FOR SEARCHING A NODE

Suppose START is the address of the first node in the linked list and DATA is the information to be searched.
After searching, if the DATA is found, POS will contain the corresponding position in the list.

SINGLY LINKED LIST

ALGORITHM FOR SEARCHING A NODE

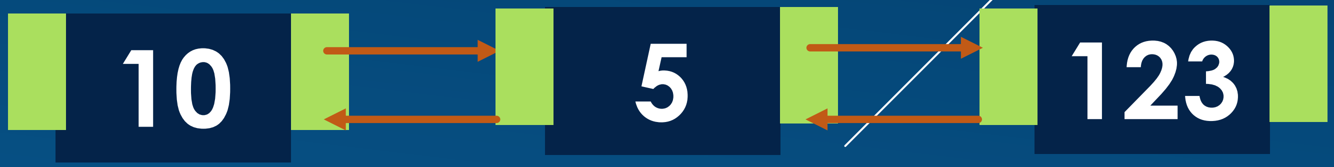
```
1. Input the DATA to be searched
2. Initialize TEMP = START; POSITION =1;
3. Repeat the step 4, 5 and 6 until (TEMP is equal to NULL)
4. If (TEMP → DATA is equal to DATA)
        (a) Display "The data is found at POSITION"
        (b) Exit
5. TEMP = TEMP → Next
6. POSITION = POSITION + 1
7. If (TEMP is equal to NULL)
        (a) Display "The data is not found in the list"
8. Exit
```

DOUBLY LINKED LIST

One in which all nodes are linked together by multiple links which help in accessing both the **successor** (next) and **predecessor** (previous) node for any arbitrary node within the list.

Every nodes in the doubly linked list has three fields:

a) Left Pointer
b) Right Pointer
c) Data



DOUBLY LINKED LIST

ALGORITHM FOR INSERTING THE NODE

Suppose **START** is the first position in linked list.

Let **DATA** be the element to be inserted in the new node.

POS is the position where the NewNode is to be inserted.

TEMP is a temporary pointer to hold the node address.

DOUBLY LINKED LIST

ALGORITHM FOR INSERTING THE NODE

```
1. Input the DATA and POS
2. Initialize TEMP = START; i = 0
3. Repeat the step 4 if (i less than POS) and (TEMP is not equal to NULL)
4. TEMP = TEMP → RightPointer;          i = i +1
5. If (TEMP not equal to NULL) and (i equal to POS)
        (a) Create a New Node
        (b) NewNode → DATA              = DATA
        (c) NewNode → RightPointer      = TEMP → RightPointer
        (d) NewNode → LeftPointer       = TEMP
        (e) (TEMP → RightPointer) → LeftPointer = NewNode
        (f ) TEMP → RightPointer        = New Node
6. Else
        (a) Display "Position NOT found"
7. Exit
```

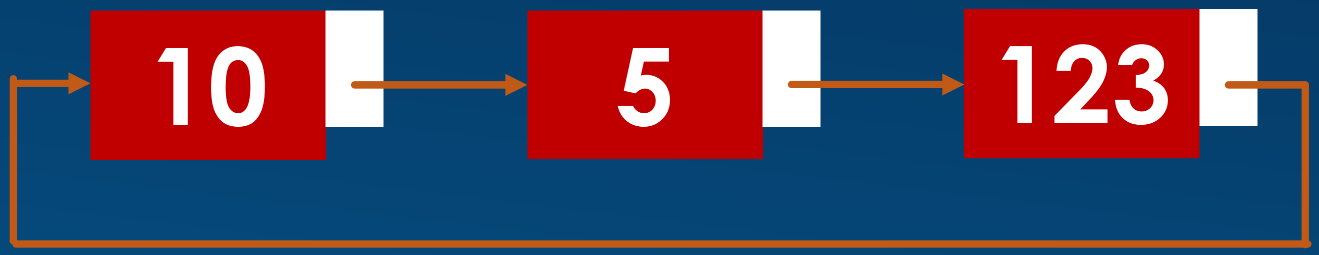
DOUBLY LINKED LIST

DISADVANTAGES OF DOUBLY LINKED LIST

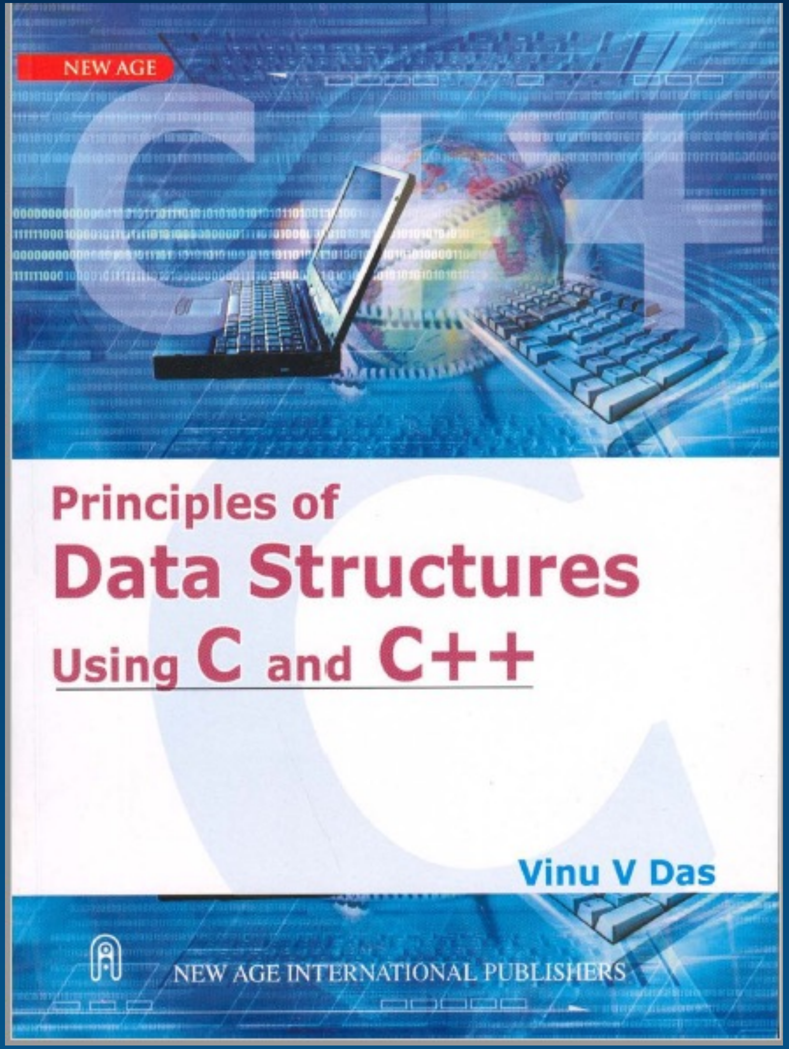
1. It consume more memory space.
2. There is a large pointer adjustment during insertion and deletion of element.
3. It consumes more time for few basic list operations.

CIRCULAR LINKED LIST

- which has no beginning and no end. A singly linked list can be made a circular linked list by simply storing the address of the very first node in the linked field of the last node.



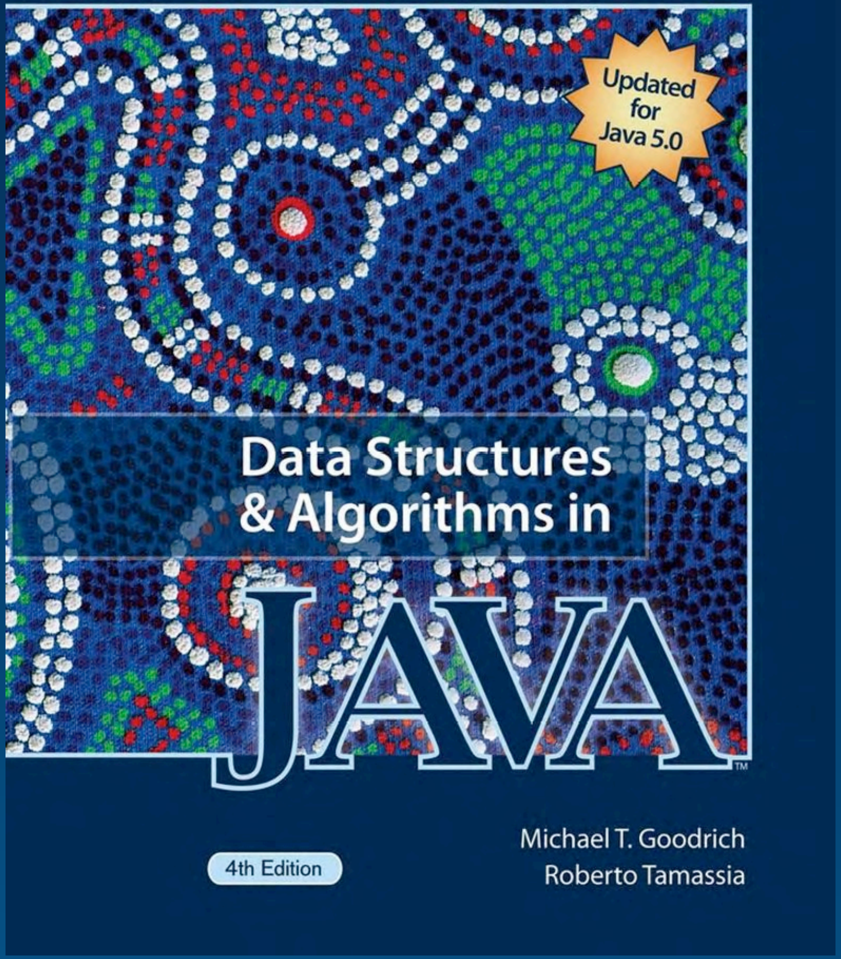
REFERENCES


NEW AGE

Principles of Data Structures Using C and C++

Vinu V Das

NEW AGE INTERNATIONAL PUBLISHERS


Updated for Java 5.0

Data Structures & Algorithms in JAVA

4th Edition

Michael T. Goodrich

Roberto Tamassia