		UNIT-V			
10	a)	Compare binary trees and binary search trees with respect to structure and operations.	L4	CO4	5 M
	b)	Explain the concept of hashing and its importance in data structures.	L2	CO3	5 M
		OR			
11	a)	How does recursion help in tree traversal? Explain with an example.	L2	CO3	5 M
	b)	Illustrate different collision resolution techniques in hashing with proper examples.	L4	CO4	5 M

25

PVP 23

Code: 23CS3201, 23IT3201, 23AM3201, 23DS3201

I B.Tech - II Semester – Regular / Supplementary Examinations
MAY 2025

DATA STRUCTURES
(Common for CSE, IT, AIML, DS)

Duration: 3 hours Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place.

BL – Blooms Level CO – Course Outcome

PART – A

		BL	CO
1.a)	Define a linear data structure.	L1	CO1
1.b)	State the worst-case time complexity of Bubble Sort.	L1	CO1
1.c)	How is the last node of a circular linked list identified?	L2	CO1
1.d)	Why does a doubly linked list require more memory than a singly linked list?	L2	CO1
1.e)	Why is stack implementation using linked lists more memory-efficient than arrays?	L2	CO1
1.f)	Give an example of a balanced and an unbalanced parenthesis expression.	L2	CO1
1.g)	What are the basic operations performed on a queue?	L1	CO1

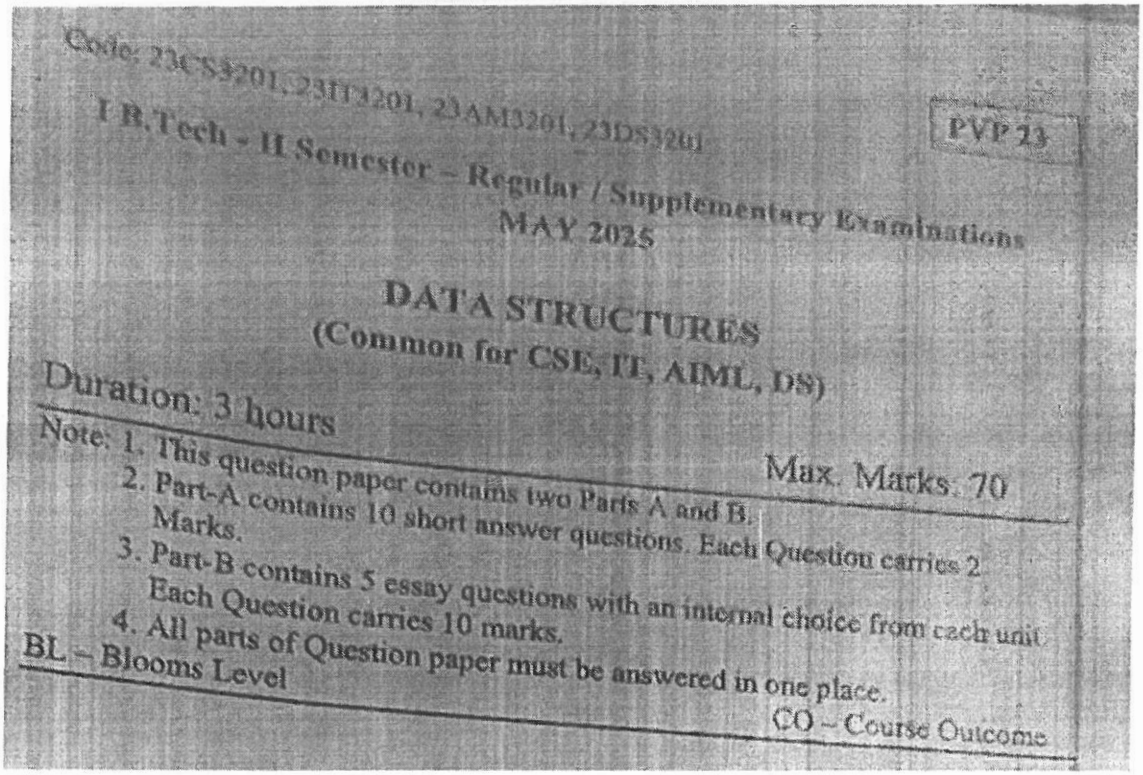
1.h)	What problem does a circular queue solve that a simple queue does not?	L2	CO1
1.i)	What is the base condition for a recursive tree traversal function?	L2	CO1
1.j)	What is a binary search tree (BST)?	L1	CO1

PART – B

			BL	CO	Max. Marks
		UNIT-I			
2	a)	Describe the importance of analyzing time and space complexities in algorithm design.	L2	CO1	5 M
	b)	Explain Abstract Data Types (ADTs) with an example.	L2	CO1	5 M
		OR			
3	a)	Compare and contrast Linear and Binary Search with suitable examples.	L2	CO2	5 M
	b)	Write and explain the algorithm for Binary Search.	L3	CO2	5 M
		UNIT-II			
4	a)	Describe the structure of a doubly linked list and compare it with a singly linked list.	L2	CO1	5 M
	b)	Compare arrays and linked lists based on memory usage, access time, and operations.	L2	CO3	5 M
		OR			

5	a)	Develop an algorithm to reverse a singly linked list and explain it.	L3	CO3	5 M
	b)	Explain how linked lists are used in dynamic memory allocation.	L2	CO3	5 M
		UNIT-III			
6	a)	Explain the push and pop operations in a stack with an example.	L3	CO2	5 M
	b)	Convert the following infix expression to postfix expression. $a + b * c/(d * e + f\hat{}g) - h * k$	L3	CO4	5 M
		OR			
7	a)	How does a stack help in checking balanced parentheses? Explain with an example.	L3	CO3	5 M
	b)	Construct a C/C++/Python program to implement a stack using an array.	L3	CO3	5 M
		UNIT-IV			
8	a)	Explain the enqueue and dequeue operations in a queue with an example.	L2	CO3	5 M
	b)	Illustrate the working of a circular queue with an example.	L3	CO3	5 M
		OR			
9	a)	Analyze the time complexity of various queue operations in both array and linked list implementation.	L4	CO4	5 M
	b)	Develop pseudo codes for enqueue, dequeue operations in a queue with linked list implementation.	L3	CO3	5 M

Scheme of evaluation



Code: 23CS3201, 23IT3201, 23AM3201, 23DS3201

PVP 23

I B.Tech - II Semester – Regular / Supplementary Examinations
MAY 2025

DATA STRUCTURES
(Common for CSE, IT, AIML, DS)

Duration: 3 hours Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place.

BL – Blooms Level CO – Course Outcome

Part A

1.a) Define a linear data structure. (2M)

Definition - 2M

1.b) State the worst-case time complexity of Bubble sort. (2M)

Time complexity - 2M

1.c) How is the last node of a circular linked list identified? (2M)

Property of last node or diagram which represents the property – 2M

1.d) Why does a doubly linked list require more memory than a singly linked list? (2M)

Any statement or diagram which states that DLL require 2 pointers while SLL require 1 pointer - 2M

1.e) Why is stack implementation using linked list more memory efficient than arrays? (2M)

Any statement which states that linked list implementation of stacks allocated memory dynamically. -2M

1.f) Give an example of a balanced and an unbalanced parenthesis expression. (2M)

Any one example of a balanced parenthesis expression - 1M

Any one example of unbalanced parenthesis expression - 1M

1.g) What are the basic operations performed on a queue? (2M)

Any two operations -2M

1.h) What problem does a Circular queue solve that a Simple queue does not? (2M)

Any statement or diagram which states that circular queue solves the problem of false overflow. -2M

1.i) What is the base condition for a recursive tree traversal function? (2M)

Any statement which means that root is NULL-2M

1.j) What is a binary search tree (BST)? (2M)

Definition – 2M

.....

Part B

Unit -1

2. a) Describe the importance of analyzing time and space complexities in algorithm design. (5M)

Importance – 2M
Definitions or examples – 3M

2.b) Explain Abstract Data Types (ADTs) with an example. (5M)

ADT definition – 1M
Examples-2M
Explanation-2M

OR

3. a) Compare and contrast Linear and Binary Search with suitable examples. (5M)

Any 5 differences or explanation of linear search and binary search – 5M

3. b) Write and explain the algorithm for Binary Search. (5M)

Algorithm/Pseudo code– 4 M
Explanation-1M

.....

Unit -2

4.a) Describe the structure of a doubly linked list and compare it with a singly linked list. (5M)

structure of a doubly linked list – 2M
structure of a singly linked list-2M
Comparison-1M
Or
Any 5 differences-5M

4.b) Compare arrays and linked lists based on memory usage, access time, and operations. (5M)

3 differences – 5M

OR

5.a) Develop an algorithm to reverse a singly linked list and explain it (5M)

Algorithm/Pseudo code– 4M
Explanation -1M

5. b) Explain how linked lists are used in dynamic memory allocation (5M)

Correct explanation- 5M

.....

Unit -3

6.a) Explain the push and pop operations in a stack with an example(5M)

push defintion-1M

pop definition-1M

Example-3M

6.b) Convert the following infix expression to postfix expression(5M)

$$a + b * c/(d * e + f\hat{}g) - h * k$$

Correct answer-5M

Partially correct answer or correct procedure-3M

OR

7.a) How does a stack help in checking balanced parenthesis? Explain with an example. (5M)

Algorithm/Pseudo code/Explanation- 2M

Example-3M

7.b) Construct a C/C++/Python program to implement a stack using an array (5M)

Program-5M

.....

Unit -4

8.a) Explain the enqueue and dequeue operations in a queue with an example(5M)

enqueue defintion-1M

dequeue definition-1M

Example-3M

8.b) Illustrate the working of a circular queue with an example. (5M)

Explanation of enqueue and dequeue in circular queue – 5M

OR

9.a) Analyze the time complexity of various queue operations in both array and linked list implementation. (5M)

Time complexity of any 5 queue operations – 5M

9.b) Develop pseudo codes for enqueue, dequeue operations in a queue with linked list implementation. (5M)

Enqueue pseudo code – 3M

Dequeue pseudo code – 2M

.....

Unit -5

10.a) Compare binary trees and binary search trees with respect to structure and operations(5M)

Binary trees and binary search trees explanation- 3M

Comparison-2M

10.b) Explain the concept of hashing and its importance in data structures (5M)

Concept of hashing – 4 M

Importance of hashing-1M

OR

11.a) How does recursion help in tree traversal? Explain with an example (5M)

Any recursive tree traversal – 3M

Example – 2M

11.b) Illustrate different collision resolution techniques with proper examples. (5M)

Closed hashing - 3M

Open hashing- 2M

.....

Key

Code: 23CS3201, 23IT3201, 23AM3201, 23DS3201 PVP 23

I B.Tech – II Semester – Regular / Supplementary Examinations
MAY 2025

DATA STRUCTURES
(Common for CSE, IT, AIML, DS)

Duration: 3 hours Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place.

BL – Blooms Level CO – Course Outcome

Part A

1.a) Define a linear data structure. (2M)

Definition - 2M

A data structure in which elements are arranged sequentially or linearly is called linear data structure.

1.b) State the worst-case time complexity of Bubble sort. (2M)

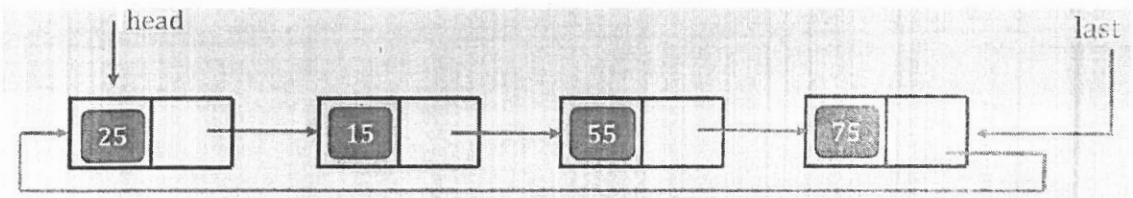
Time complexity - 2M

Worst-case time complexity of Bubble sort is $O(n^2)$ or $\theta(n^2)$

1.c) How is the last node of a circular linked list identified? (2M)

Property of last node or diagram which represents the property – 2M

The last node of a circular linked list is the node whose next pointer points to the head.



1.d) Why does a doubly linked list require more memory than a singly linked list? (2M)

Statement or diagram - 2M

A doubly linked list requires more memory because each node stores two pointers (one for the next node and one for the previous node), whereas a singly linked list only stores one pointer (for the next node).

1.e) Why is stack implementation using linked list more memory efficient than arrays? (2M)

Statement-2M

A stack implemented using a linked list is more memory efficient than an array because it dynamically allocates memory only for the elements present, avoiding the need for a pre-allocated fixed-size array that may waste space

1.f) Give an example of a balanced and an unbalanced parenthesis expression. (2M)

Any one example of a balanced parenthesis expression - 1M.
For example, [{ }]
Any one example of un balanced parenthesis expression - 1M.
For example, [{)] .

1.g) What are the basic operations performed on a queue? (2M)

Any two operations -2M

Enqueue(Q, x):	Insert a new element x at the rear end of the Queue Q.
Dequeue(Q):	Delete an element from the front end of the Queue Q.
Peek (Q):	Returns the element at the front end of the queue Q.
IsEmpty(Q):	Returns true if Q is empty and false otherwise.
IsFull(Q):	Returns true if Q is full and false otherwise.
Size(Q):	Returns the number of elements in Q
Display(Q):	Displays the elements of Q from front end to rear end

1.h) What problem does a Circular queue solve that a Simple queue does not? (2M)

Statement-2M

A circular queue solves the space wastage problem of a simple queue by reusing empty slots at the array's beginning through wraparound, avoiding false overflow.

1.i) What is the base condition for a recursive tree traversal function? (2M)

Any statement which means that root is NULL-2M

Base condition is "if(root=NULL) return"

1.j) What is a binary search tree (BST)? (2M)

Definition – 2M

A binary search tree (BST) is a binary tree that satisfies the following property.

1. The values in the non-empty left subtree of a node are smaller than the value in the node.
2. The values in the non-empty right subtree of a node are greater than the value in the node.

.....

Part B

Unit -1

2. a) Describe the importance of analyzing time and space complexities in algorithm design. (5M)

Importance – 2M
Definitions or examples – 3M

Performance of an algorithm is measured in 2 ways.

1. Time complexity 2. Space complexity

1. Time complexity:

The amount of time that is needed to execute an algorithm is called time complexity. In general, time complexity of an algorithm is measured in terms of the number of basic operations performed by the algorithm.

2. Space complexity:

The amount of memory or space that is needed to execute an algorithm is called space complexity.

The space complexity $S(P)$ of any algorithm P can be written as

$$S(P) = C + S_P(I)$$

,Where C:Constant that denotes fixed part,

$S_P(I)$:Variable part that depends on instance characteristics (I).

Analysing time and space complexities in algorithm design is crucial for evaluating an algorithm's efficiency and scalability. Understanding these complexities helps us to choose the best algorithm for a problem, balancing speed and memory needs.

.....

2.b) Explain Abstract Data Types (ADTs) with an example. (5M)

ADT definition – 1M
Examples-2M

Explanation-2M

An Abstract Data Type is a logical description of how data is organized and the operations that can be performed on that data, without specifying how those operations are implemented.
Examples: Queue, Stack, Tree, Graph, Hash Table

1. Queue: Queue is a linear data structure that follows the FIFO (First In, First Out) principle. This means that the element that is added to the queue first is the one that gets removed first.

2. Stack: Stack is a linear data structure that follows the LIFO (Last In, First Out) principle. This means that the last element that is added to the stack is the one that gets removed first.

3. Tree: Tree is a hierarchical data structure that consists of nodes connected by edges. It represents a collection of elements organized in a tree-like structure, where each node has a parent and possibly multiple children, except for the root node, which has no parent.

4. Graph: Graph is a collection of nodes and edges, where each edge connects two nodes. The graph can be used to represent various relationships between objects.

5. Hash Table: Hash Table is a data structure that stores data in an associative manner using a key-value pairs. It uses a hash function to map keys to specific locations (or buckets) in hash table.

.....

OR

3. a) Compare and contrast Linear and Binary Search with suitable examples. (5M)

Any 5 differences or explanation of linear search and binary search – 5M

Linear Search sequentially checks each element in the list until the target is found or the list ends. It works on both sorted and unsorted arrays.

Example:

Consider the following list of elements

40 20 60 70 50 80 90 10

Supposer the search element $x = 70$

Initially, $i = 0$.

Compare x with a[0]

$x \neq a[0] = 40$

Increment i

Now $i = 1$.

Compare x with a[1]

$x \neq a[1] = 20$

Increment i

Now $i = 2$.

Compare x with a[2]

$x \neq a[2] = 60$

Increment i

Now $i = 3$.

Compare x with a[3]

$x = a[3] = 70$

Element found at index $i = 3$. Return i.

Binary Search, in contrast, requires a sorted array and works by repeatedly dividing the search interval in half.

Let us consider the array

10 15 20 25 35 40 50 55 60 65 70

1. Suppose the search element is 55

Here $n = 11$, $x = 55$

low	high	mid	
1	11	$\frac{1+11}{2} = 6$	$x > a[mid]$ Go to right sub list
7	11	$\frac{7+11}{2} = 9$	$x < a[mid]$ Go to left sub list
7	8	$\frac{7+8}{2} = 7$	$x > a[mid]$ Go to right sub list
8	8	$\frac{8+8}{2} = 8$	$x = a[mid]$ Element found

Linear Search	Binary Search
Linear search can be applied to both sorted and unsorted lists	Binary search can be applied to sorted lists only
Linear search is simple and straight forward to implement	Binary search is complex as compared to linear search
Linear search is slow process	Binary search is fast process
More number of comparisons are required	Less number of comparisons are required
Time Complexity: Best Case: $\Theta(1)$ Worst Case: $\Theta(n)$ Average Case: $\Theta(n)$	Time Complexity: Best Case: $\Theta(1)$ Worst Case: $\Theta(\log n)$ Average Case: $\Theta(\log n)$

.....

3. b) Write and explain the algorithm for Binary Search. (5M)

Algorithm/Pseudo code– 4 M
Explanation-1M

Iterative version:

```
//a: Sorted array of size n
//x: Element to be searched
BinarySearch(a, n, x)
{
    low=1;  high=n;
    while(low<=high)
    {
        mid = ⌊(low+high)/2⌋;
        if(x=a[mid])
            return(mid);
        else if(x<a[mid])
            high=mid-1;
        else
            low=mid+1;
    }
    return(-1);
}
```

Recursive version:

```
//a: Sorted Array of size n
//x: Element to be searched
//In main function, we call RBinarySearch(a,1,n,x)
RBinarySearch(a, low, high, x)
{
    if(low<=high)
    {
        mid=(low + high)/2;
        if(x=a[mid])
            return(mid);
        else if(x<a[mid])
            return RBinarySearch(a, low, mid-1, x);
        else
            return RBinarySearch(a, mid+1, high, x);
    }
    return(-1);
}
```

Element x is compared with $a[mid]$

1. If $x = a[mid]$, then return mid.
2. If $x < a[mid]$, then search the left sub list.
3. If $x > a[mid]$, then search the right sub list.

.....

Unit -2

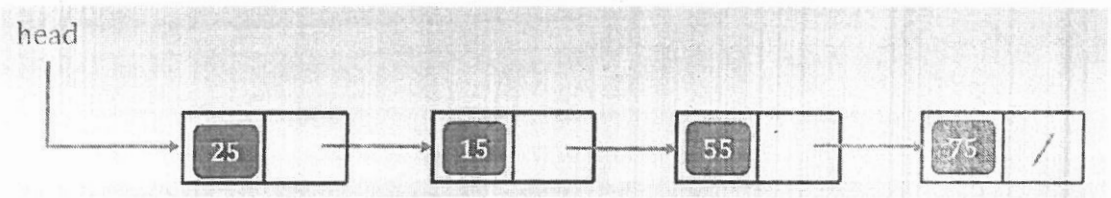
4.a) Describe the structure of a doubly linked list and compare it with a singly linked list. (5M)

structure of a doubly linked list – 2M
structure of a singly linked list-2M
Comparison-1M
Or
Any 5 differences-5M

Singly linked list:

In a singly linked list, each node(except the last node) contains only one link which points to the subsequent node in the list. The link portion of the last node contains the value NULL

Example:



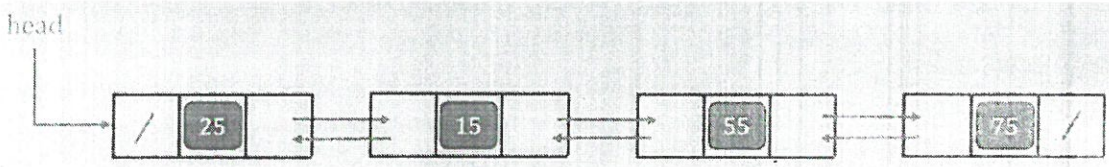
Node structure:

```
struct node
{
    int data;
    struct node *next;
};
```

Doubly linked list:

In a doubly linked list, each node (except first and last nodes) contains two links, one points to the next node and other points to the previous node.

Example:



Node structure:

```
struct node
{
    struct node *prev;
    int data;
    struct node *next;
};
```

Singly Linked List	Doubly Linked List
Each node points only to the next node.	Each node points to both the next and previous nodes.
Node contains data and a next pointer.	Node contains data, a next pointer and a prev pointer
In SLL, we can traverse only in one direction (forward).	In DLL, we can traverse in both direction (forward and backward).
Requires less memory	Requires more memory
Less flexible due to single-direction traversal.	More flexible due to bidirectional traversal.

.....

4.b) Compare arrays and linked lists based on memory usage, access time, and operations. (5M)

3 differences – 5M

Array	Linked List
Memory Allocation : Contiguous memory allocation.	**Memory Allocation**: Non-contiguous memory allocation
Size Flexibility: Fixed size (cannot be resized dynamically).	**Size Flexibility**: Dynamic size (can grow or shrink during runtime).
Storage Type: Static data structure.	**Storage Type**: Dynamic data structure.
Access Time: Fast ($O(1)$ access using index).	**Access Time**: Slow ($O(n)$ access, requires traversal from the head).
Implementation: Simple to implement and use.	**Implementation**: More complex due to pointer management.

.....

OR

5.a) Develop an algorithm to reverse a singly linked list and explain it (5M)

Algorithm/Pseudo code– 4M

Explanation -1M

Iterative version:

```
ReverseList(head)
{
    if(head==NULL or head->next==NULL)
        return(head);
    curr=head; prev=NULL;
    while(curr!=NULL)
    {
        nxt=curr->next;
        curr->next=prev;
        prev=curr;
        curr=nxt;
    }
    return(prev);
}
```

Time Complexity: $\Theta(n)$

Initialize three pointers **prev** as NULL, **curr** as **head**, and **next** as NULL.

Iterate through the linked list. In a loop, do the following:

- Store the next node, **next = curr -> next**
- Update the next pointer of curr to prev, **curr -> next = prev**
- Update prev as curr and curr as next, **prev = curr** and **curr = next**

Recursive version:

```
ReverseList(head)
{
    if(head==NULL or head->next==NULL)
        return(head);
    rest=ReverseList(head->next);
    head->next->next=head;
    head->next=NULL;
    return(rest);
}
```

Time Complexity: $\Theta(n)$

.....

5. b) Explain how linked lists are used in dynamic memory allocation (5M)

Correct explanation- 5M

- Linked lists play a key role in dynamic memory allocation by managing memory blocks efficiently in systems like operating systems or memory allocators (e.g., malloc in C). They are used to track free and allocated memory blocks in a heap, enabling flexible memory management.
- In dynamic memory allocation, the heap is divided into blocks of memory, some free and some allocated. A linked list of free blocks (often called a free list) is maintained, where each node in the list represents a free memory block. Each node contains the block's starting address, size, and a pointer to the next free block. For example, if the heap has free blocks at addresses 100 (size 50) and 200 (size 30), the free list might look like: [100, 50] -> [200, 30] -> NULL.
- When a program requests memory (e.g., via malloc), the allocator traverses the free list to find a suitable block (using strategies like first-fit or best-fit). If a block of size 40 is needed, the allocator might select the block at 100, allocate 40 bytes, and update the free list to [140, 10] -> [200, 30] -> NULL (splitting the block). If no suitable block is found, the allocator may request more memory from the OS.
- When memory is freed (e.g., via free), the block is added back to the free list, and adjacent free blocks are merged to reduce fragmentation. For instance, freeing a block at 140 (size 10) next to 150 (size 20) merges them into [140, 30]. Linked lists enable this dynamic resizing and merging because they allow easy insertion, deletion, and traversal of memory blocks without requiring contiguous memory, unlike arrays. This flexibility makes linked lists ideal for managing the unpredictable nature of dynamic memory allocation in systems programming.

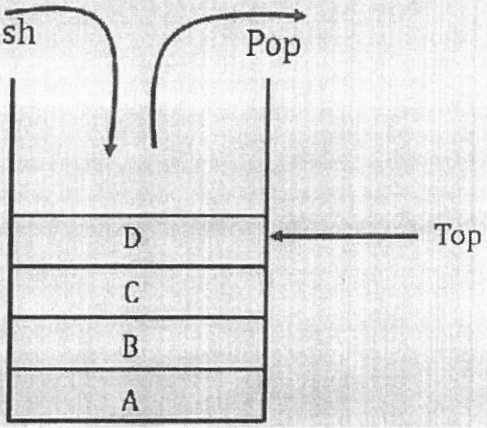
.....

Unit -3

6.a) Explain the push and pop operations in a stack with an example(5M)

push defintion-1M
pop definition-1M
Example-3M

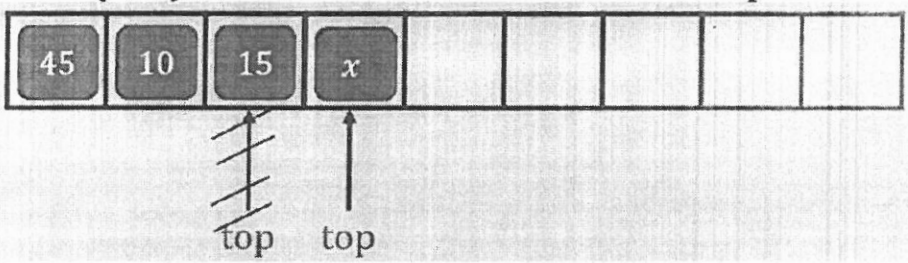
A stack is a data structure that follows the Last In, First Out (LIFO) principle. This means that the last element added to the stack is the first one to be removed.



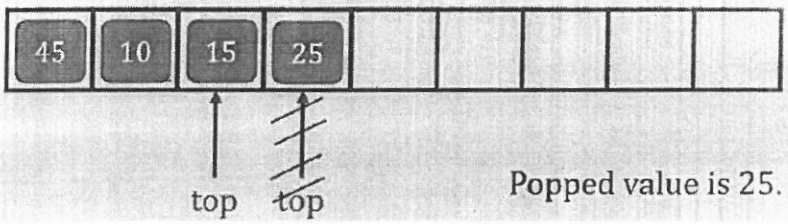
Push (S, x): Insert element x at the top of stack S.

Pop (S): Deletes the top element of the stack

1. **Push(s, x):** Adds the element x at the top of the stack s.



2. **Pop(s):** Removes and returns the top element from the stack s.



.....

6.b) Convert the following infix expression to postfix expression(5M)

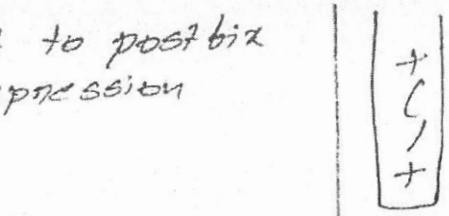
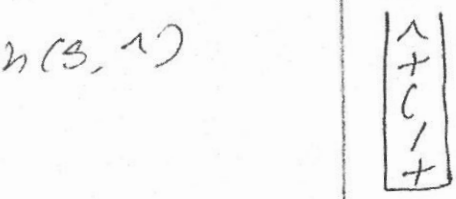
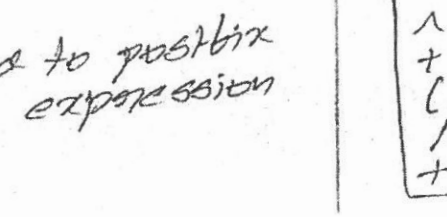
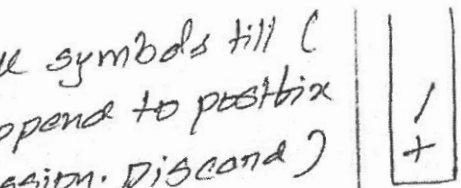
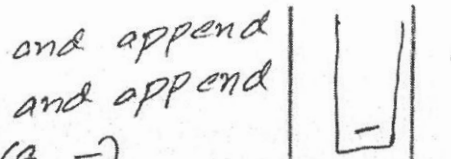
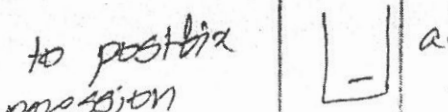
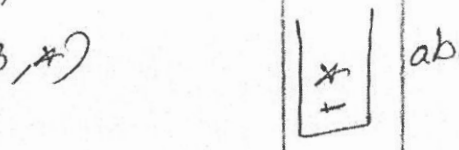
$a + b * c/(d * e + f \hat{} g) - h * k$

Correct answer-5M
Partially correct answer or correct procedure-3M

Given infix expression is

a+b*c/(d*e+b^g)-h*k

symbol scanned	Action	stack	postfix expression
—	—	⌊⌋	—
a	Append to postfix expression	⌊⌋	a
+	push(S, +)	⌊+⌋	a
b	Append to postfix expression	⌊+⌋	ab
*	push(S, *)	⌊*/+⌋	ab
c	Append to postfix expression	⌊*/+⌋	abc
/	pop(S) and append push(S, /)	⌊//+⌋	abc*
(	push(S, ( )	⌊(/+⌋	abc*
d	append to postfix expression	⌊(/+⌋	abc*d
*	push(S, *)	⌊*(/+⌋	abc*d
e	append to postfix expression	⌊*(/+⌋	abc*de
+	pop(S) and append push(S, +)	⌊+(/+⌋	abc*de*

b	append to postfix expression		abc*de*b
^	push(S, ^)		abc*de*b
g	append to postfix expression		abc*de*bg
)	pop all symbols till (and append to postfix expression. Discard)		abc*de*bg^+
-	pop(S) and append pop(S) and append push(S, -)		abc*de*bg^+/+
h	append to postfix expression		abc*de*bg^+/+h
*	push(S, *)		abc*de*bg^+/+h

k	append to postfix expression	* -	abc*de*fg^+/+hk
-	pop all symbols and append to postfix expression		abc*de*fg^+/+hk*-

∴ postfix expression is abc*de*fg^+/+hk*-

.....

OR

7.a) How does a stack help in checking balanced parenthesis? Explain with an example. (5M)

Algorithm/Pseudo code/Explanation– 2M
Example-3M

1. Scan symbols of expression from left to right.
2. If symbol is left parenthesis,
   Push on to the stack.
3. If symbol is right parenthesis
   If stack empty
   Invalid.
   else
   Pop symbol from the stack.
   If popped parenthesis does not match the parenthesis being scanned
   Invalid: mismatched parenthesis.
4. After scanning all the symbols
   If stack is empty
   Valid: Balanced parenthesis.
   else
   Invalid.

Example:

Given sequence of parenthesis is [{ ( )]

symbol scanned	Action	stack
–	–	⊔
[	push(S, [ )	[
{	push(S, {)	{ [
}	pop(S) '{' matched with '}'	[
(	push(S, ( )	([
)	pop(S) '(' matched with ')'	[
]	pop(S) '[' matched with ']'	⊔

∴ All symbols are scanned and stack is empty.
So, the given parenthesis is balanced

.....

7.b) Construct a C/C++/Python program to implement a stack using an array (5M)

Program-5M

```
//Program to implement stack using array
#include<stdio.h>
#include<stdlib.h>
#define n 22
int top=-1;
void Push(int s[], int x);
int Pop(int s[]);
int Peek(int s[]);
void Display(int s[]);
void IsEmpty();
void IsFull();
int Length();
void main()
{
  int s[n],x,l,ch;
  while(1)
```

```
  {
    printf("\n1.Push    2. Pop    3. Peek    4. Display");
    printf("\n5.Is Empty  6. Is Full  7. Length   8. exit  \n");
    printf("Enter your choice:");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1:
        printf("Enter data:");
        scanf("%d",&x);
        Push(s,x);
        Display(s);
        break;
      case 2:
        x=Pop(s);
        if(x!=-1)
          printf("Popped element is %d",x);
        Display(s);
        break;
      case 3:
        x=Peek(s);
        if(x!=-1)
          printf("Element at top is %d",x);
        Display(s);
        break;
      case 4:
        Display(s);
        break;
      case 5:
        IsEmpty();
        break;
      case 6:
        IsFull();
        break;
      case 7:
        l=Length();
        printf("Number of elements in stack is %d \n",l);
        break;
      case 8:
        exit(0);
      default:
        printf("Enter correct choice");
    }
  }
}
void Push(int s[], int x)
{
  if(top==n-1)
  {
```

```
        printf("Stack overflow \n");
        return;
    }
    top=top+1;
    s[top]=x;
}
int Pop(int s[])
{
    int x;
    if(top==-1)
    {
        printf("Stack underflow \n");
        return(-1);
    }
    x=s[top];
    top=top-1;
    return(x);
}
int Peek(int s[])
{
    int x;
    if(top==-1)
    {
        printf("Stack underflow \n");
        return(-1);
    }
    x=s[top];
    return(x);
}
void Display(int s[])
{
    int i;
    printf("\nElements in stack: \n");
    for(i=top;i>=0;i--)
        printf("%d \n",s[i]);
}
void IsEmpty()
{
    if(top==-1)
        printf("Stack is empty \n");
    else
        printf("Stack is not empty \n");
}
void IsFull()
{
    if(top==n-1)
        printf("Stack is full \n");
    else
        printf("Stack is not full \n");
```

```
}
int Length()
{
  return(top+1);
}
```

.....

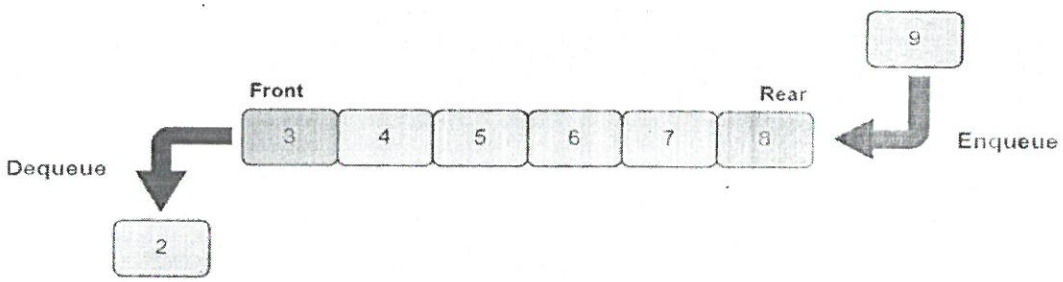
Unit -4

8.a) Explain the enqueue and dequeue operations in a queue with an example(5M)

enqueue defintion-1M
dequeue definition-1M
Example-3M

A queue is a restricted linear list in which all insertions and deletions can be done at opposite ends, called the **rear** and **front** respectively.

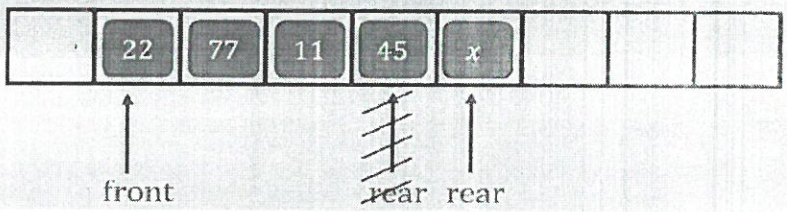
A queue is a data structure that follows the First In, First Out (FIFO) principle. This means that the first element added to the queue is the first one to be removed.



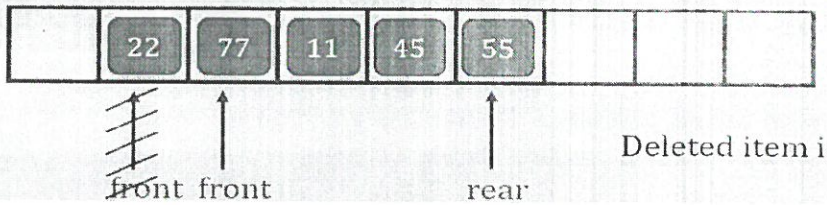
Enqueue(Q, x): Insert a new element x at the rear end of the Queue Q.

Dequeue(Q): Delete an element from the front end of the Queue Q.

1. **Enqueue (Q, x):** Insert a new element x at the rear end of the Queue Q.



2. **Dequeue (Q):** Delete an element from the front end of the Queue Q.

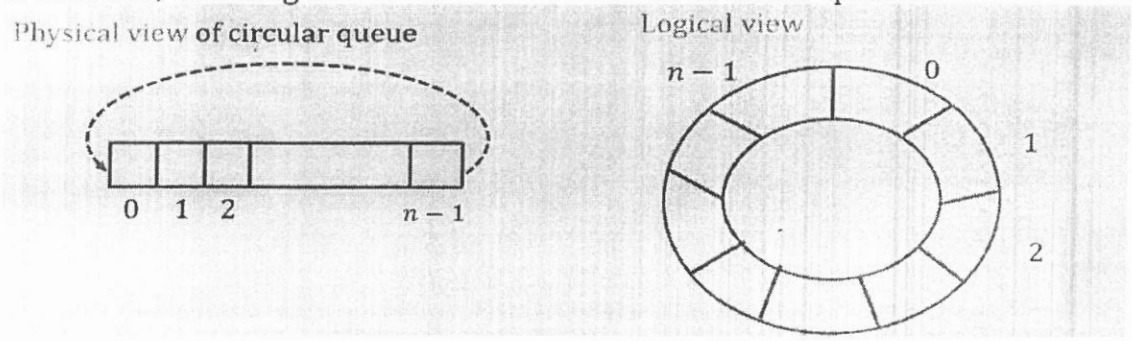


.....

8.b) Illustrate the working of a circular queue with an example. (5M)

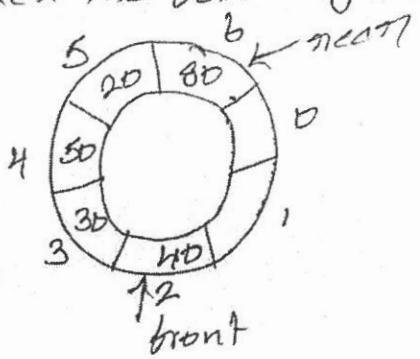
Explanation of enqueue and dequeue in circular queue – 5M

A circular queue is a fixed-size queue where the last position connects back to the first, allowing efficient reuse of empty spaces after dequeues. It prevents memory wastage by wrapping around when the end is reached, avoiding the "false overflow" issue of linear queues.



Suppose $n = 7$

Let us consider the following circular queue



now rear = n-1, To insert a element we do rear = (rear+1) % n =

Here we get rear = 0.

we insert new element 0th position

Similarly, for dequeue operation.

we do front = (front+1) % n

.....

OR

9.a) Analyze the time complexity of various queue operations in both array and linked list implementation. (5M)

Time complexity of any 5 queue operations – 5M

Operation	Array (Simple Queue)	Linked List
Enqueue	$O(1)$	$O(1)$
Dequeue	$O(1)$	$O(1)$
Front/Peek	$O(1)$	$O(1)$
IsEmpty	$O(1)$	$O(1)$
IsFull	$O(1)$	Not applicable

In linked list implementation, for dequeue operation, if we assume that we are maintain tail pointer to last node. If we don't maintain tail pointer, it will be $O(n)$

.....

9.b) Develop pseudo codes for enqueue, dequeue operations in a queue with linked list implementation. (5M)

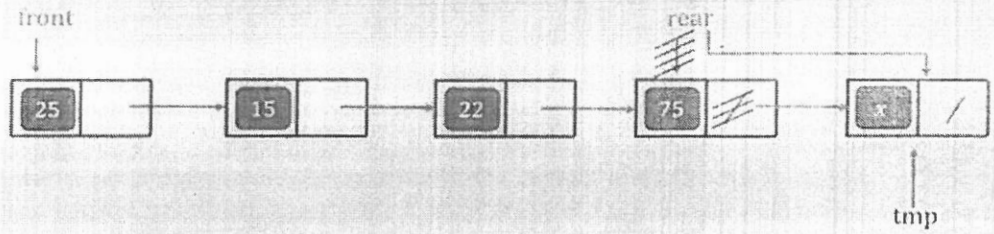
Enqueue pseudo code – 3M
Dequeue pseudo code – 2M

Assumptions for the functions:

```
typedef struct node Node;
struct node
{
    int data;
    Node *next;
};
Node *front=NULL,*rear=NULL;
```

- ***front*** is a pointer to the first node of the linked list.
- ***rear*** is a pointer to the last node of the linked list.
- For **enqueue** operation, we add a node at the end of the linked list.
- For **dequeue** operation, we delete a node at the beginning of the linked list.

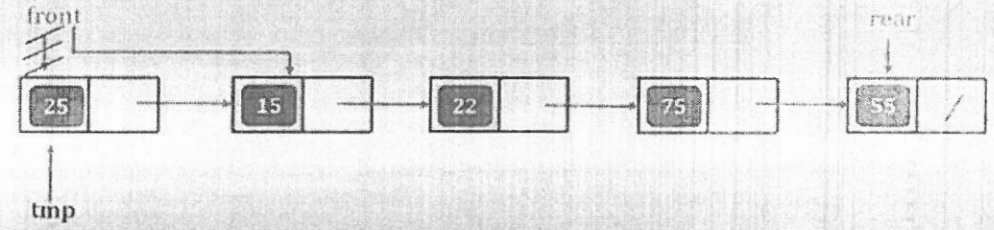
1. **Enqueue (x):** Insert a new element x at the rear end of the Queue.




```
void Enqueue(int x)
{
    tmp=(Node*)malloc(sizeof(Node));
    tmp->data = x;
    tmp->next=NULL;
    if(front==NULL)  /*If Queue is empty*/
        front=tmp;
    else
        rear->next=tmp;
    rear=tmp;
}
```

Time Complexity: $\Theta(1)$

2. **Dequeue ():** Delete an element from the front end of the Queue.




```
int Dequeue()
{
    if(front==NULL)
    {
        printf("Queue is empty \n");
        return(-1);
    }
   x=front->data;
   tmp=front;
   if(front==rear)
       front=rear=NULL;
   else
       front=front->next;
   free(tmp);
   return(x);
}
```

Time Complexity: $\Theta(1)$

.....

Unit -5

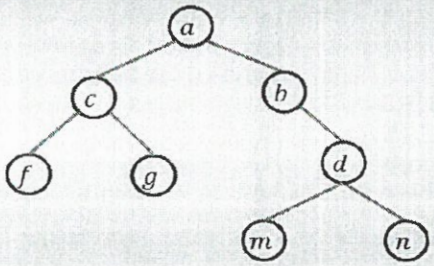
10.a) Compare binary trees and binary search trees with respect to structure and operations(5M)

Binary trees and binary search trees explanation– 3M
Comparison-2M

Binary tree:

A binary tree is a hierarchical data structure in which every node has maximum 2 children, called the left child and right child.

Example:



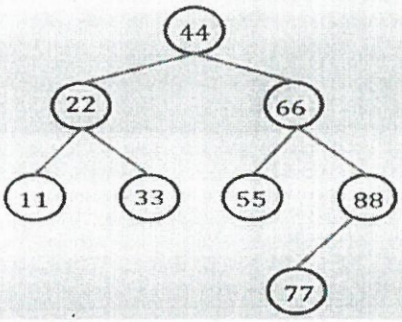
Binary search tree:

A binary search tree (BST) is a binary tree that has a value associated with each of its nodes.

The values satisfy Binary Search Tree Property:

1. The values in the non-empty left subtree of a node are smaller than the value in the node.
2. The values in the non-empty right subtree of a node are greater than the value in the node.

Example:



1. Structure:
   - Binary Tree (BT):
     - No ordering constraints—nodes can be arranged in any way.
     - A node can have at most two children (left and right).
   - Binary Search Tree (BST):
     - Follows a strict ordering property:
       - Left child ≤ Parent
       - Parent < Right child
     - A node can have at most two children (left and right).

2. Search Operation:
   - BT:
     - No guaranteed order → Requires O(n) time (worst-case traversal).
   - BST:

- ➢ if search value x is less than root value, we continue search in left subtree
- ➢ if search value x is greater than root value, we continue search in right subtree

3. Insertion & Deletion:
   - BT:
     - ➢ No fixed rules—insertion/deletion depends on application (e.g., heaps have their own rules).
   - BST:
     - ➢ Must maintain order
       - Left child ≤ Parent
       - Parent < Right child

.....

10.b) Explain the concept of hashing and its importance in data structures (5M)

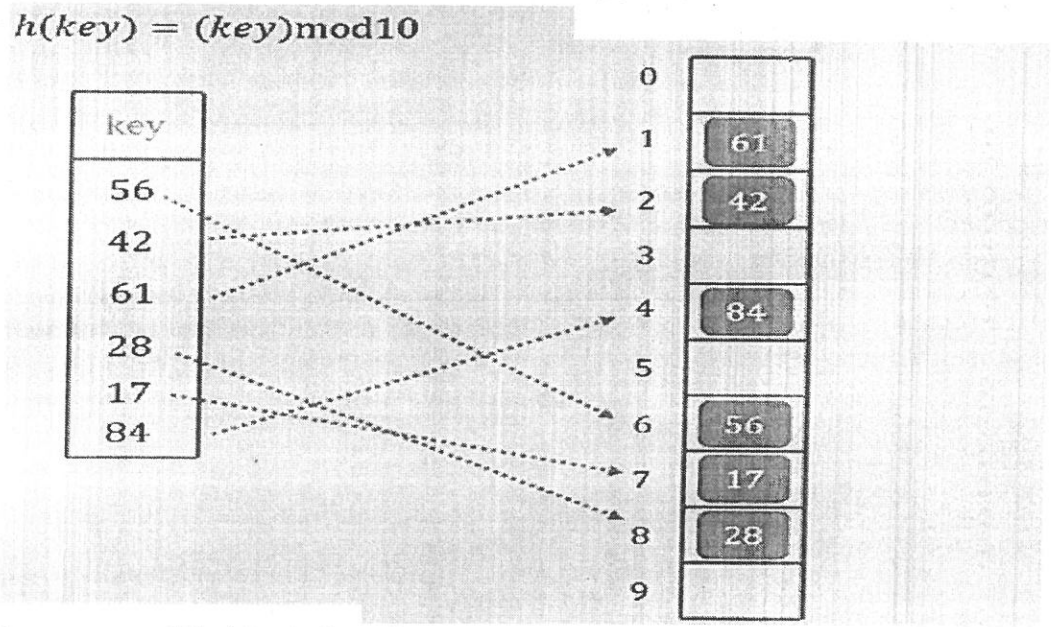
Concept of hashing – 4 M
Importance of hashing-1M

Hash table is a data structure in which keys are mapped to array positions by a hash function.

In hash table, an element with key k, is stored at index $h(k)$, where h is a hash function.

For accessing any element, we apply the same hash function to the key and then access the element at the address given by hash function.

The process of generating addresses from keys is called hashing.



Importance of Hashing in Data Structures:

1. Fast Data Access – Enables O(1) average-time lookups, insertions, and deletions using hash tables.
2. Efficient Storage – Minimizes memory usage by mapping large data to compact hash values.

3. **Key Applications** – Powers hash tables, databases, caches, cryptography, and duplicate detection.

.....

OR

11.a) How does recursion help in tree traversal? Explain with an example (5M)

Any recursive tree traversal – 3M
Example – 2M

We use recursion to travers a binary tree. For example, inorder traversal can be done as follows:

Inorder traversal is a traversal method that visits nodes in the following order.

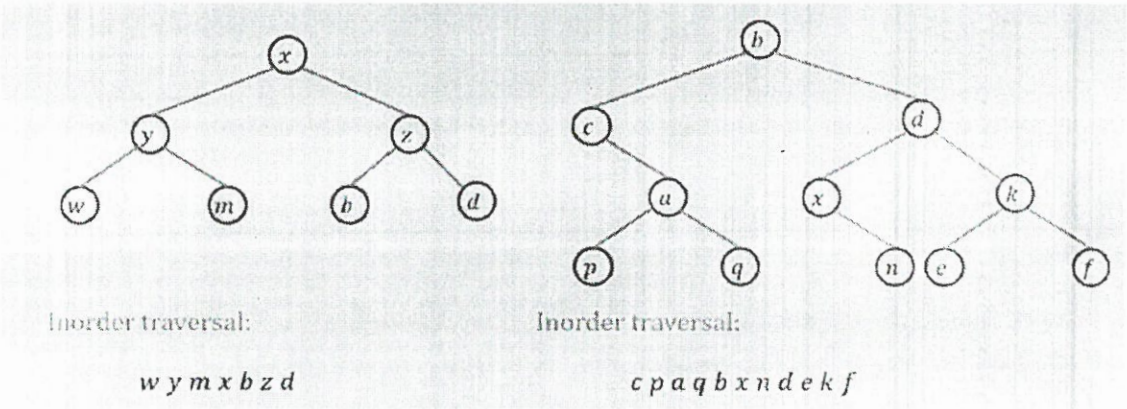
1. **Traverse the left subtree** in Inorder.
2. **Visit the root node.**
3. Traverse the **right subtree** in Inorder.

Pseudo code:

```
void InOrder(Node *root)
{
    if(root)
    {
        InOrder(root->lchild);
        printf("%d ",root->data);
        InOrder(root->rchild);
    }
}
```

Time Complexity: $O(n)$

Example:



Inorder traversal:

w y m x b z d

Inorder traversal:

c p a q b x n d e k f

NOTE: Consider explanation of any traversal technique

.....

11.b) Illustrate different collision resolution techniques with proper examples. (5M)

Closed hashing - 3M
Open hashing- 2M

A collision occurs when a hash function maps two different keys to same location in a hash table. A key mapped to an already occupied table location results in collision.