Data Structure Using C (313301)

Model Question Paper — Set C · Model Answers

Q.1

a) Data structure and classification. 2 M

Data structure: A way of organizing and storing data in memory so that it can be used efficiently.

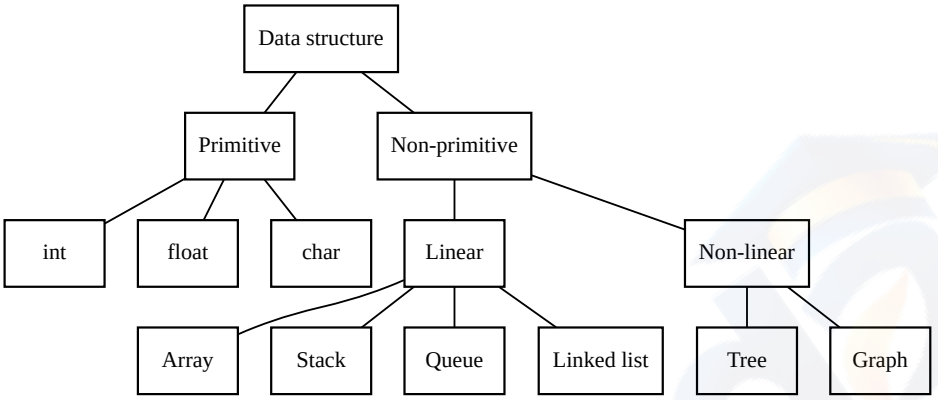


Fig.: Classification of data structures

(Definition 1 M, classification 1 M)

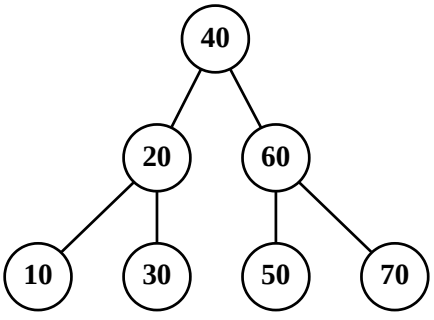
b) Stack vs queue. 2 M

Stack	Queue
Works on LIFO (Last In First Out).	Works on FIFO (First In First Out).
Insertion and deletion at one end (TOP).	Insertion at REAR, deletion at FRONT.
Operations: PUSH, POP.	Operations: INSERT (enqueue), DELETE (dequeue).
Example: pile of plates.	Example: people in a ticket line.

(Any two points, 1 M each)

c) Binary search tree. 2 M

A **binary search tree (BST)** is a binary tree in which, for every node, all values in its **left** sub-tree are **smaller** than the node and all values in its **right** sub-tree are **greater** than the node. Both sub-trees are also BSTs.



Example BST

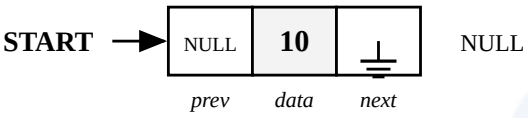
d) Advantages of circular queue. 2 M

1. Empty locations freed by deletion are reused, so memory is used efficiently.
2. “Queue full” occurs only when all locations are really full (no false overflow).
3. FRONT and REAR move circularly, so no shifting of elements is needed.

(Any two, 1 M each)

e) Node declaration of doubly linked list. 2 M

```
struct node
{
    struct node *prev;   /* address of previous node */
    int data;            /* information */
    struct node *next;   /* address of next node */
};
```



One node of a doubly linked list

f) Level and in-degree. 2 M

i) Level of a node: The root is at level 0; the level of any other node is one more than the level of its parent.

ii) In-degree of a node: The number of edges coming into a node. In a tree, in-degree of the root is 0 and of every other node is 1.

g) Searching and its methods. 2 M

Searching is the process of finding the location of a given element (key) in a list of elements.

Methods: 1) Linear (sequential) search 2) Binary search.

Q.2

a) Sorting techniques; selection sort {33, 11, 44, 22}. 4 M

Sorting techniques: Bubble sort, Selection sort, Insertion sort, Quick sort, Merge sort, Radix sort.

Selection sort: In every pass the smallest element of the unsorted part is found and swapped with the first unsorted position.

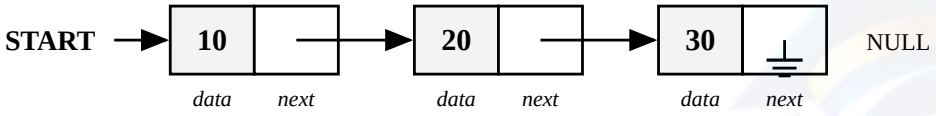
Pass	Smallest selected	Array after pass
Initial	—	33, 11, 44, 22
1	11	11, 33, 44, 22
2	22	11, 22, 44, 33
3	33	11, 22, 33, 44

Sorted list: 11, 22, 33, 44

(Techniques 1 M, explanation 1 M, example 2 M)

b) Operations on singly linked list. 4 M

1. **Creation** — creating the list by allocating nodes and linking them.
2. **Traversal / display** — visiting each node from START till NULL.
3. **Insertion** — at beginning, at end, or after a given node.
4. **Deletion** — from beginning, from end, or a given node.
5. **Searching** — finding a node with a given value.
6. **Counting / reversing / concatenation** of lists.



Singly linked list

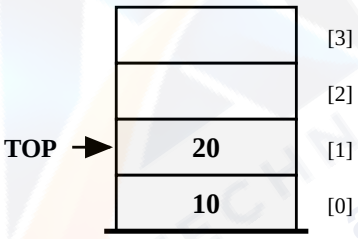
(Any four operations, 1 M each)

c) PUSH and POP using array. 4 M

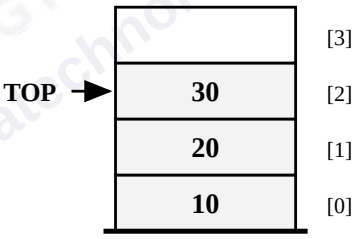
Stack is stored in array *stack[MAX]*; variable **TOP** holds the index of the top element (TOP = −1 means empty).

PUSH(x): if TOP = MAX − 1 → overflow; else TOP = TOP + 1 and stack[TOP] = x.

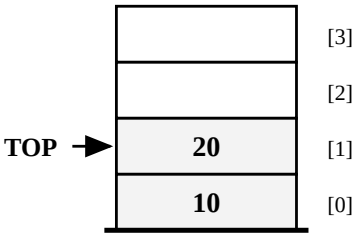
POP(): if TOP = −1 → underflow; else x = stack[TOP] and TOP = TOP − 1.



Initial (TOP = 1)



PUSH 30 (TOP = 2)



POP → 30 (TOP = 1)

(PUSH 2 M, POP 2 M)

d) Expression tree for (a + b * c) / (d − e ^ f) 4 M

'/' is applied last, so it is the root. In (a + b * c), * has higher precedence, so it is lower in the tree; in (d − e ^ f), ^ is lower.

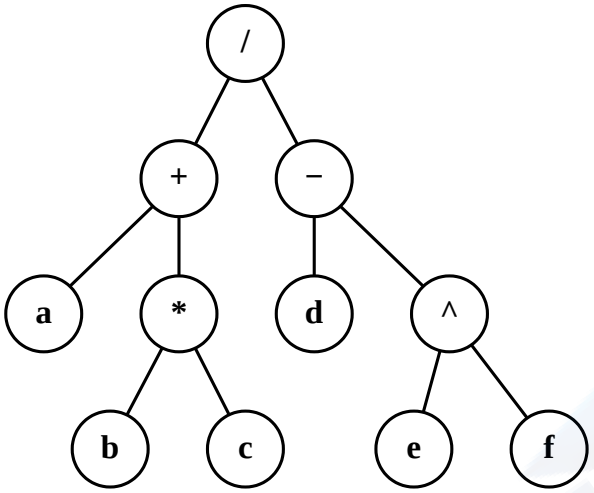


Fig.: Expression tree

(Correct tree 4 M)

Q.3

a) C program — bubble sort. 4 M

```
#include <stdio.h>

int main()
{
    int a[20], n, i, j, temp;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for (i = 0; i < n - 1; i++)          /* passes */
    {
        for (j = 0; j < n - 1 - i; j++)    /* compare adjacent */
        {
            if (a[j] > a[j + 1])
            {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
    printf("Sorted array: ");
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

Output:

```
Enter number of elements: 5
Enter 5 elements:
42 17 63 8 29
Sorted array: 8 17 29 42 63
```

b) Infix to postfix: ((P + Q) * R) ^ (S − T) 4 M

Symbol	Stack	Postfix expression
(	(	
(	((	
P	((	P
+	((+	P
Q	((+	PQ
)	(	PQ+
*	(*	PQ+
R	(*	PQ+R
)	—	PQ+R*
^	^	PQ+R*
(	^(	PQ+R*
S	^(	PQ+R*S
-	^(-	PQ+R*S
T	^(-	PQ+R*ST
)	^	PQ+R*ST-
(end)	—	PQ+R*ST-^

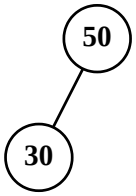
Postfix expression: PQ+R*ST-^

c) Heap; max heap for 30, 50, 20, 60, 10. 4 M

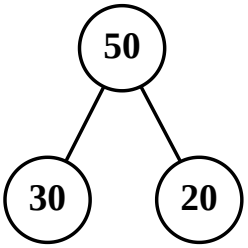
Heap is a complete binary tree that satisfies the heap property. In a **max heap** every parent is greater than or equal to its children, so the largest element is at the root. (In a min heap every parent is smaller.)

Construction: insert each element at the next free position (level-wise, left to right) and move it up (swap with parent) while it is greater than its parent.

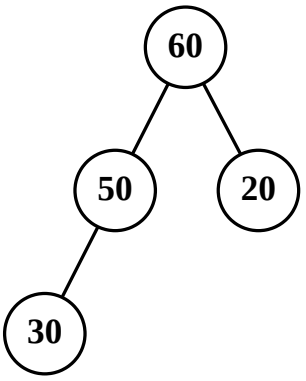
insert 30



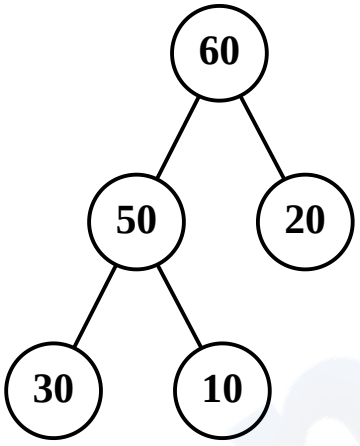
insert 50 (swap 50 with 30)



insert 20



insert 60 (swap 60 with 30, swap 60 with 50)



insert 10

Final max heap (array form): 60, 50, 20, 30, 10

(Explanation 1 M, construction 3 M)

d) Static vs dynamic memory allocation.

4 M

Static memory allocation	Dynamic memory allocation
Memory allocated at compile time.	Memory allocated at run time.
Size fixed, cannot be changed.	Size can be changed (grow / shrink).
Memory may be wasted if not fully used.	No wastage; allocated as needed and released with free().
Uses stack area of memory.	Uses heap area of memory.
Faster, no allocation overhead.	Slightly slower due to allocation at run time.
Example: int a[50];	Example: p = (int*) malloc(n * sizeof(int));

(Any four points, 1 M each)

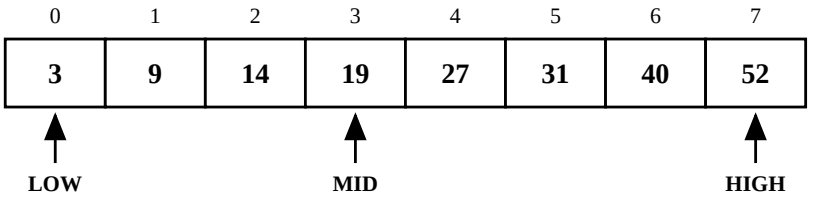
Q.4

a) Binary search for 19.

4 M

Array is sorted. Key = 19, LOW = 0, HIGH = 7.

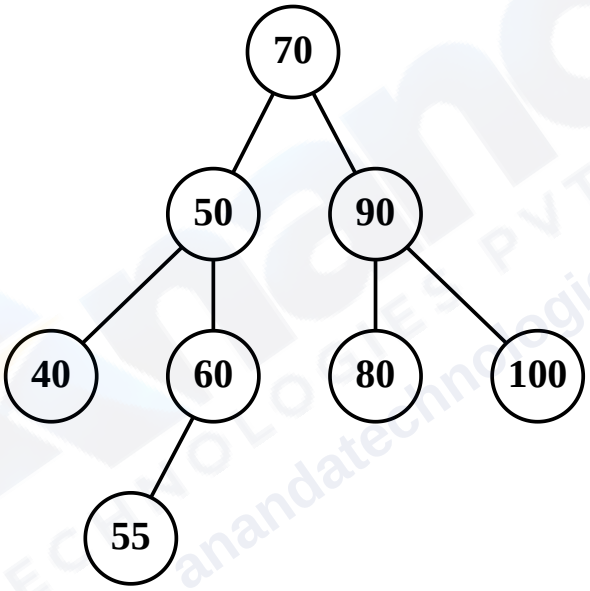
Step 1: LOW = 0, HIGH = 7, MID = (0 + 7) / 2 = 3; A[3] = 19 → found



Element 19 is found at index 3, i.e. position 4.

b) BST for 70, 50, 90, 40, 60, 80, 100, 55; preorder. 4 M

1. **70**: root
2. **50**: 50 < 70 → left of 70
3. **90**: 90 > 70 → right of 70
4. **40**: 40 < 70 , 40 < 50 → left of 50
5. **60**: 60 < 70 , 60 > 50 → right of 50
6. **80**: 80 > 70 , 80 < 90 → left of 90
7. **100**: 100 > 70 , 100 > 90 → right of 90
8. **55**: 55 < 70 , 55 > 50 , 55 < 60 → left of 60



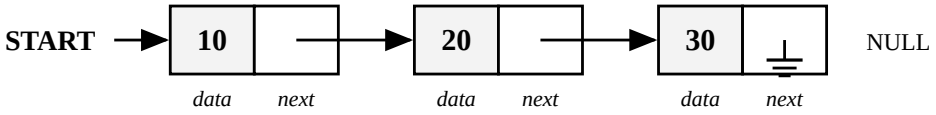
Binary search tree

Preorder (Root, Left, Right): 70, 50, 40, 60, 55, 90, 80, 100

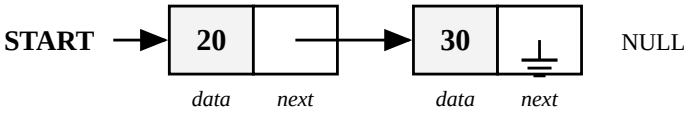
(BST 3 M, preorder 1 M)

c) Algorithm — delete node from beginning of SLL. 4 M

1. Start.
2. If START = NULL, print “Underflow / list empty” and stop.
3. Set TEMP = START.
4. Set START = START → next (second node becomes first).
5. Free(TEMP).
6. Stop.

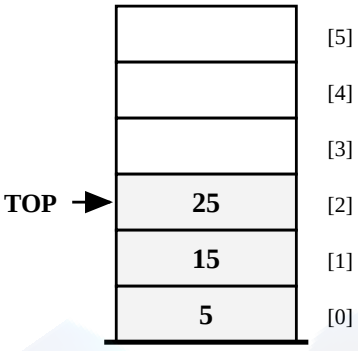


Before deletion

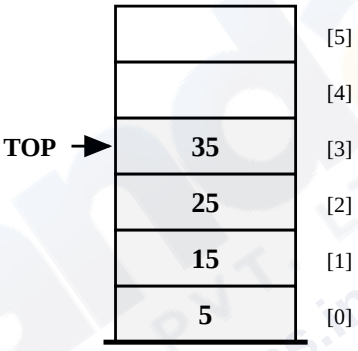


After deleting first node (10)

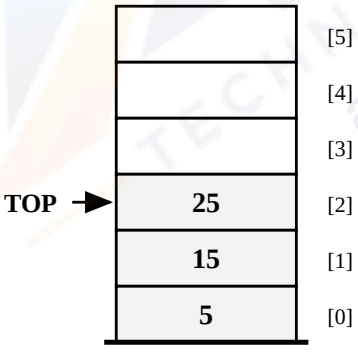
d) Effect of PUSH / POP on stack of size 6. 4 M



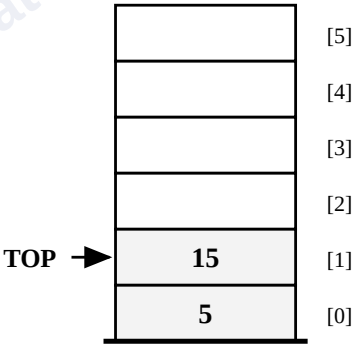
Initial (TOP = 2)



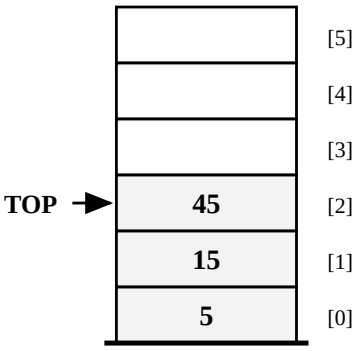
i) PUSH 35



ii) POP → 35



iii) POP → 25

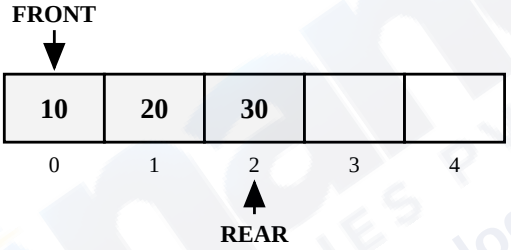


iv) PUSH 45

Final stack: 5, 15, 45 (45 at TOP, TOP = 2).

e) Types of queue. 4 M

1. **Linear (simple) queue:** insertion at REAR, deletion at FRONT (FIFO).
2. **Circular queue:** last position is connected to the first, so free locations are reused.
3. **Priority queue:** each element has a priority; highest priority element is deleted first.
4. **Double ended queue (Deque):** insertion and deletion at both ends (input-restricted and output-restricted deque).



Linear queue

(Four types with explanation, 1 M each)

Q.5

a) C program — circular queue. 6 M

```
#include <stdio.h>
#define MAX 5

int cq[MAX];
int front = -1, rear = -1;

void insert(int x)
{
    if ((rear + 1) % MAX == front)
    {
        printf("Circular Queue Overflow\n");
        return;
    }
    if (front == -1)              /* first element */
        front = 0;
```

```
    rear = (rear + 1) % MAX;
    cq[rear] = x;
    printf("%d inserted\n", x);
}

void delete()
{
    if (front == -1)
    {
        printf("Circular Queue Underflow\n");
        return;
    }
    printf("%d deleted\n", cq[front]);
    if (front == rear)            /* only one element was there */
        front = rear = -1;
    else
        front = (front + 1) % MAX;
}

void display()
{
    int i;
    if (front == -1)
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue: ");
    i = front;
    while (1)
    {
        printf("%d ", cq[i]);
        if (i == rear)
            break;
        i = (i + 1) % MAX;
    }
    printf("\n");
}

int main()
{
    insert(10);
    insert(20);
    insert(30);
    insert(40);
    insert(50);
    insert(60);
    display();
    delete();
    delete();
    insert(60);
    insert(70);
    display();
    return 0;
}
```

Output:

```
10 inserted
20 inserted
30 inserted
40 inserted
50 inserted
Circular Queue Overflow
Queue: 10 20 30 40 50
10 deleted
20 deleted
60 inserted
70 inserted
Queue: 30 40 50 60 70
```

Note: after two deletions, 60 and 70 are stored at index 0 and 1 (REAR wrapped around) — the advantage of circular queue.

b) Insertion sort: {52, 18, 39, 7, 64, 25} 6 M

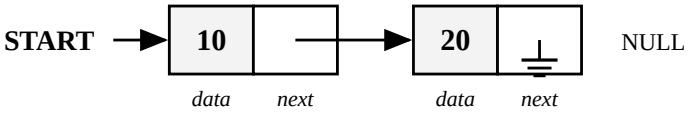
Each element (key) is inserted into its correct position in the sorted left part by shifting greater elements right.

Iteration	Key	Array after iteration
Initial	—	52, 18, 39, 7, 64, 25
1	18	18, 52, 39, 7, 64, 25
2	39	18, 39, 52, 7, 64, 25
3	7	7, 18, 39, 52, 64, 25
4	64	7, 18, 39, 52, 64, 25
5	25	7, 18, 25, 39, 52, 64

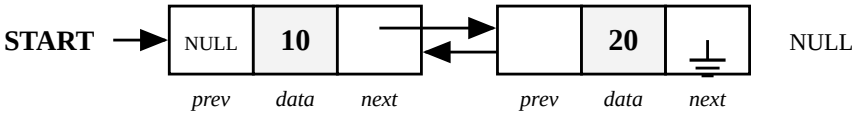
Sorted list: 7, 18, 25, 39, 52, 64

c) Singly vs doubly vs circular linked list. 6 M

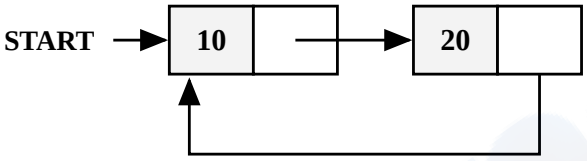
Point	Singly linked list	Doubly linked list	Circular linked list
Node parts	data, next	prev, data, next	data, next (last node points to first)
Traversal	Forward only	Forward and backward	Forward, from any node, round and round
Last node	next = NULL	next = NULL	next = address of first node
Memory per node	Less (one pointer)	More (two pointers)	Same as singly
Delete a given node	Needs previous node (traverse)	Direct using prev pointer	Needs previous node (traverse)
Use	Simple lists, stacks	Navigation (back/forward), deque	Round-robin scheduling, buffers



Singly



Doubly

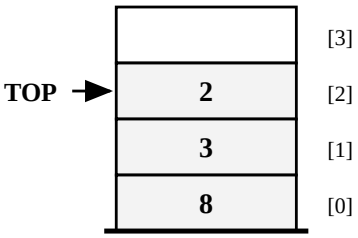


Circular

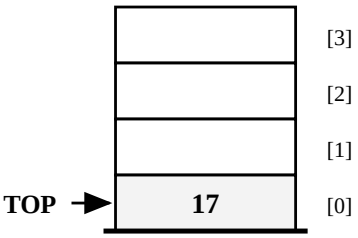
Q.6

a) Evaluate postfix: 8, 3, 2, ^, +, 6, 2, /, − 6 M

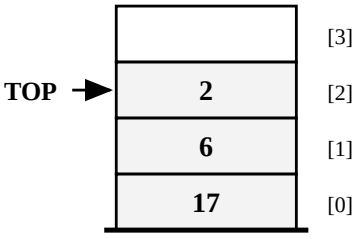
Symbol	Operation	Stack (bottom → top)
8	Push 8	8
3	Push 3	8, 3
2	Push 2	8, 3, 2
^	Pop 2, 3; 3 ^ 2 = 9; push 9	8, 9
+	Pop 9, 8; 8 + 9 = 17; push 17	17
6	Push 6	17, 6
2	Push 2	17, 6, 2
/	Pop 2, 6; 6 / 2 = 3; push 3	17, 3
-	Pop 3, 17; 17 - 3 = 14; push 14	14



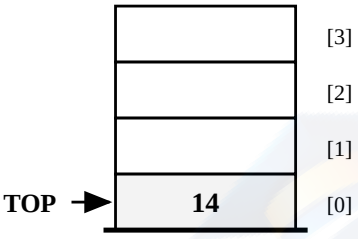
after 8, 3, 2



after ^, +



after 6, 2

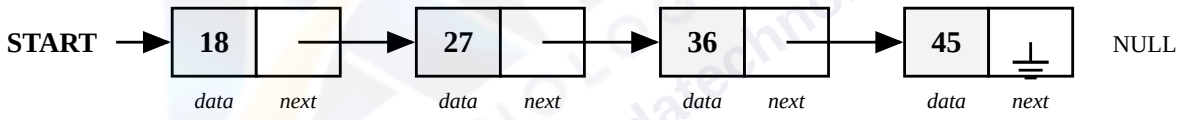


final

Result = 14

b) Create SLL 18, 27, 36, 45 and insert 30 after 27. 6 M

Step 1 — Creation: nodes 18, 27, 36, 45 are created and linked; START points to 18; last node next = NULL.

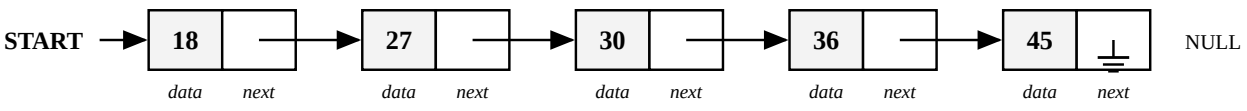


List after creation

Step 2 — Find node 27: PTR = START (18) → 18 ≠ 27, PTR = PTR→next → 27 found.

Step 3 — Create new node: NEW→data = 30.

Step 4 — Link: NEW→next = PTR→next (address of 36); PTR→next = NEW.



List after inserting 30 after 27

```
new = (struct node*) malloc(sizeof(struct node));
new->data = 30;
new->next = ptr->next;   /* 30 -> 36 */
ptr->next = new;         /* 27 -> 30 */
```

(Creation 2 M, insertion steps with diagrams 4 M)

c) Tree terms for Fig. 1. 6 M

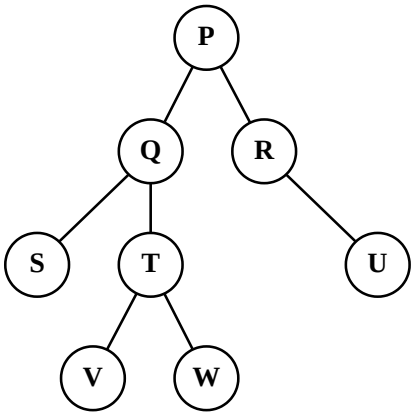


Fig. 1

1. **Leaf nodes:** S, V, W, U
2. **Degree of node Q:** 2 (children S and T)
3. **Level of node V:** 3 (P = 0, Q = 1, T = 2, V = 3)
4. **Height of tree:** 3 (longest path P → Q → T → V)
5. **Siblings of S:** T (same parent Q)
6. **Inorder traversal:** S, Q, V, T, W, P, R, U

(1 M each)

Note: This is a **model question paper prepared by Ananda Technologies** only for practice and exam preparation, based on the pattern of previous MSBTE question papers and the K-scheme syllabus. It is **not an official MSBTE paper**, and it does **not** mean that these questions will come in the exam. Answers follow standard textbook points; marks shown are indicative. Free for students — anandatechnologies.in