Data Structure Using C (313301)

Model Question Paper — Set B · Model Answers

Q.1

a) Operations on data structure. 2 M

1. **Traversing** — visiting each element once.
2. **Inserting** — adding a new element.
3. **Deleting** — removing an element.
4. **Searching** — finding the location of an element.
5. **Sorting** — arranging elements in order.
6. **Merging** — combining two lists.

(Any four, ½ M each)

b) Static vs dynamic memory allocation. 2 M

Static memory allocation	Dynamic memory allocation
Memory is allocated at compile time.	Memory is allocated at run time.
Size is fixed; cannot change while running.	Size can grow or shrink as needed.
Memory may be wasted or may fall short.	No wastage; allocated only when required.
Example: int a[10];	Example: malloc(), calloc(), free().

(Any two points, 1 M each)

c) Applications of stack. 2 M

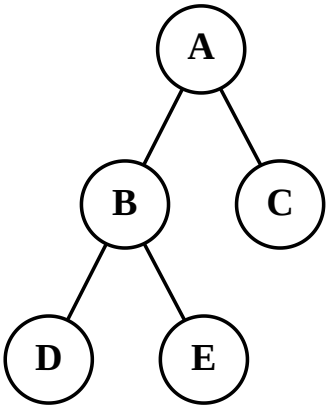
1. Conversion of infix expression to postfix / prefix.
2. Evaluation of postfix expression.
3. Recursion (function calls are stored on stack).
4. Reversing a string / list.
5. Undo operation in editors, back button in browser.

(Any two, 1 M each)

d) Sibling and depth of tree. 2 M

i) Sibling: Nodes having the same parent are called siblings.

ii) Depth of tree: The maximum level of any node in the tree, i.e. the length of the longest path from the root to a leaf (height of tree).



B and C are siblings; D and E are siblings. Depth = 2 (levels 0, 1, 2)

e) Applications of queue. 2 M

1. CPU / job scheduling in operating system.
2. Printer queue (spooling).
3. Handling requests on a shared resource (server, call centre).
4. Breadth First Search (BFS) in graphs.

(Any two, 1 M each)

f) Heap and its types. 2 M

Heap: A heap is a complete binary tree in which the value of each node is related to its children by a fixed order (heap property).

Types: 1) **Max heap** — every parent ≥ its children (largest at root). 2) **Min heap** — every parent ≤ its children (smallest at root).

g) Types of binary tree traversal. 2 M

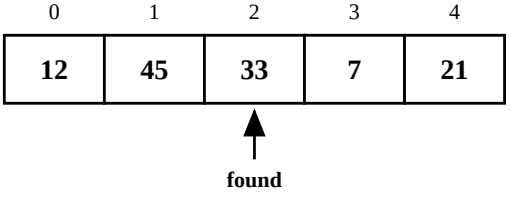
1. Inorder traversal (Left, Root, Right)
2. Preorder traversal (Root, Left, Right)
3. Postorder traversal (Left, Right, Root)

(Level-order traversal visits nodes level by level.)

Q.2

a) Linear search with example. 4 M

Working: Linear (sequential) search compares the key with each element of the list one by one, starting from the first element. If a match is found, its position is returned; if the end of the list is reached, the element is not present. Elements need not be sorted. Time complexity O(n).



Index	A[i]	Comparison
0	12	33 = 12 ? No
1	45	33 = 45 ? No
2	33	33 = 33 ? Yes — found

Element 33 is found at index 2 (position 3).

(Working 2 M, example 2 M)

b) Circular linked list. 4 M

A **circular linked list** is a linked list in which the next pointer of the last node points back to the first node instead of NULL. So the list forms a circle and has no NULL at the end.

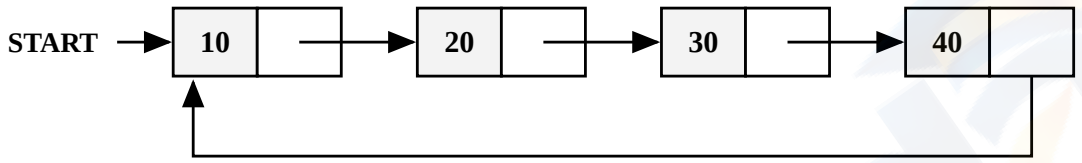


Fig.: Circular singly linked list

Advantages over linear linked list:

1. Any node can be reached from any other node by moving forward.
2. No NULL at the end; traversal can start from any node.
3. Useful for round-robin scheduling and circular buffers.
4. Insertion at the end is easy when a pointer to the last node is kept.

(Description 1 M, diagram 1 M, advantages 2 M)

c) Priority queue. 4 M

A **priority queue** is a queue in which every element has a **priority**. The element with the highest priority is deleted first. Elements with the same priority are processed in FCFS (first come first served) order.

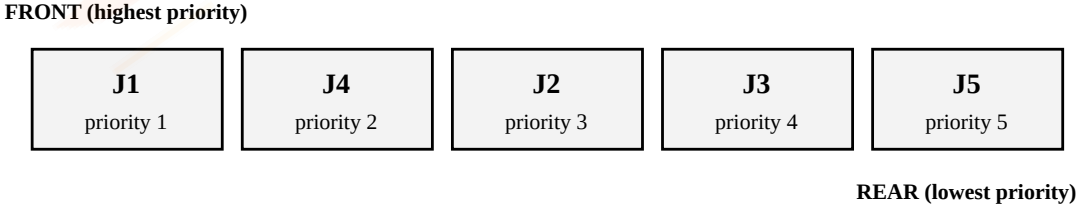


Fig.: Priority queue (1 = highest priority)

- **Insertion:** the element is placed at its position according to its priority.
- **Deletion:** the element with the highest priority (at front) is removed.
- **Types:** ascending priority queue and descending priority queue.
- **Applications:** CPU scheduling, interrupt handling, Dijkstra's shortest path.

(Explanation 2 M, sketch 2 M)

d) Expression tree for (2x + 3y)² − (a − 5b) / c 4 M

'−' is applied last (lowest precedence, outside brackets), so it is the root. Its left sub-tree is (2x + 3y)^2 and right sub-tree is (a − 5b) / c.

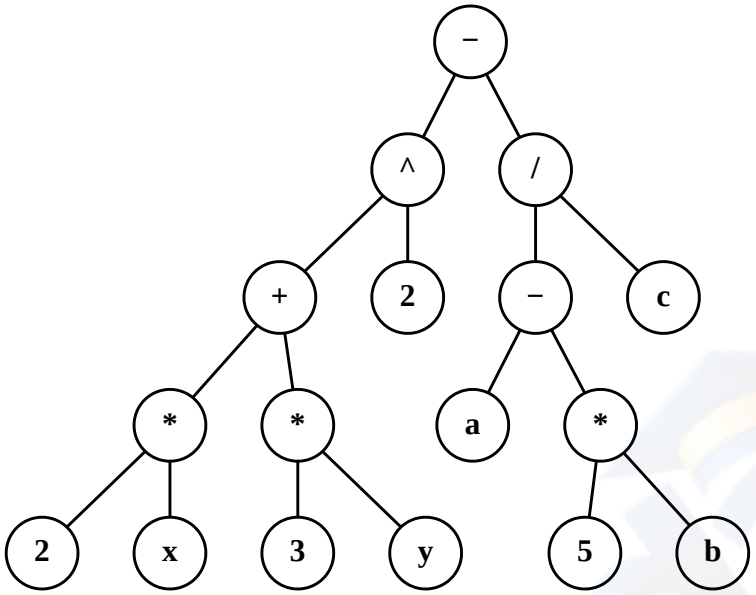


Fig.: Expression tree

(Correct tree 4 M)

Q.3

a) Algorithms for inorder and postorder traversal. 4 M

Inorder (Left, Root, Right) — INORDER(ROOT):

1. If ROOT = NULL, return.
2. INORDER(ROOT → left).
3. Visit (print) ROOT → data.
4. INORDER(ROOT → right).

Postorder (Left, Right, Root) — POSTORDER(ROOT):

1. If ROOT = NULL, return.
2. POSTORDER(ROOT → left).
3. POSTORDER(ROOT → right).
4. Visit (print) ROOT → data.

```
void inorder(struct node *r)   { if (r != NULL) { inorder(r->left);  printf("%d
", r->data); inorder(r->right); } }
void postorder(struct node *r) { if (r != NULL) { postorder(r->left);
postorder(r->right); printf("%d ", r->data); } }
```

(Each algorithm 2 M)

b) Infix to postfix: A * (B + C) − D / E 4 M

Symbol	Stack	Postfix expression
A	—	A
*	*	A
(	*(	A
B	*(	AB
+	*(+	AB
C	*(+	ABC
)	*	ABC+
-	-	ABC+*
D	-	ABC+*D
/	-/	ABC+*D
E	-/	ABC+*DE
(end)	—	ABC+*DE/-

Postfix expression: ABC+*DE/-

c) C program — create SLL (insert at end) and display. 4 M

```
#include <stdio.h>
#include <stdlib.h>

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

struct node *start = NULL;

void insertEnd(int x)
{
    struct node *temp, *p;
    temp = (struct node *)malloc(sizeof(struct node));
    temp->data = x;
    temp->next = NULL;
    if (start == NULL)          /* list empty: new node is first */
        start = temp;
    else
    {
        p = start;
        while (p->next != NULL)  /* go to last node */
            p = p->next;
        p->next = temp;
    }
}

void display()
{
```

```
    struct node *p = start;
    if (p == NULL)
    {
        printf("List is empty\n");
        return;
    }
    while (p != NULL)
    {
        printf("%d -> ", p->data);
        p = p->next;
    }
    printf("NULL\n");
}

int main()
{
    int n, i, x;
    printf("How many nodes? ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("Enter data: ");
        scanf("%d", &x);
        insertEnd(x);
    }
    display();
    return 0;
}
```

Output:

```
How many nodes? 3
Enter data: 5
Enter data: 10
Enter data: 15
5 -> 10 -> 15 -> NULL
```

d) General tree vs binary tree. 4 M

General tree	Binary tree
A node can have any number of children.	A node can have at most two children.
Sub-trees are not ordered as left and right.	Sub-trees are ordered: left child and right child.
A general tree cannot be empty (has at least a root).	A binary tree can be empty.
Traversal: preorder, postorder (no inorder).	Traversal: inorder, preorder, postorder.
Example: file-system folder tree.	Example: expression tree, BST.

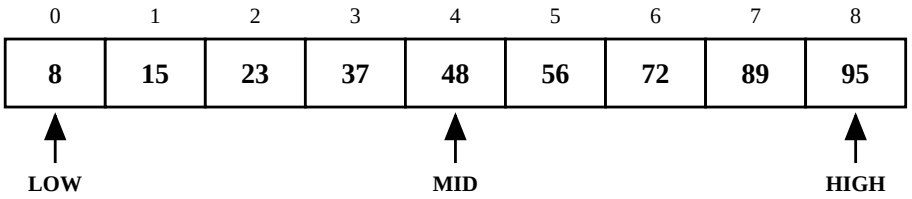
(Any four points, 1 M each)

Q.4

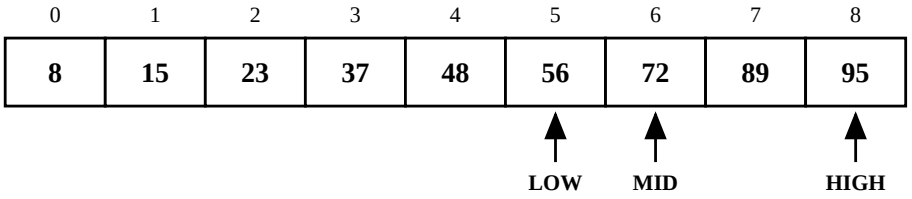
a) Binary search for 72. 4 M

Array is sorted. Key = 72, LOW = 0, HIGH = 8.

Step 1: LOW = 0, HIGH = 8, MID = (0 + 8) / 2 = 4; A[4] = 48 → 72 > 48, so LOW = MID + 1



Step 2: LOW = 5, HIGH = 8, MID = (5 + 8) / 2 = 6; A[6] = 72 → found

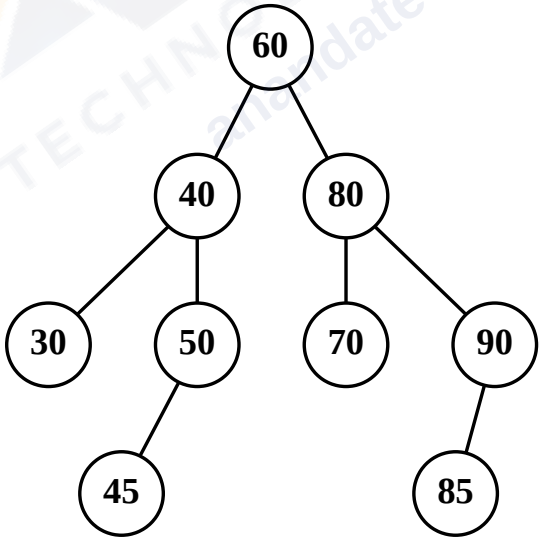


Element 72 is found at index 6, i.e. position 7.

b) BST for 60, 40, 80, 30, 50, 70, 90, 45, 85. 4 M

Smaller values go to the left, greater values go to the right.

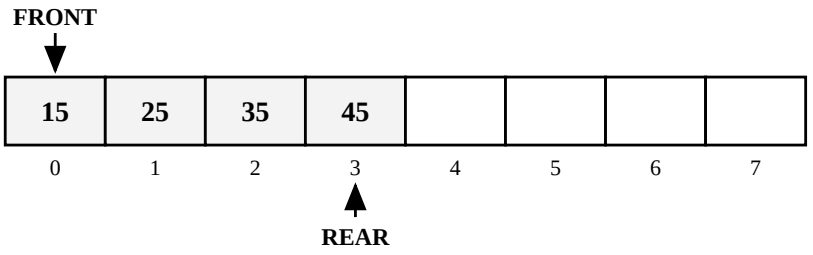
1. **60**: root
2. **40**: 40 < 60 → left of 60
3. **80**: 80 > 60 → right of 60
4. **30**: 30 < 60 , 30 < 40 → left of 40
5. **50**: 50 < 60 , 50 > 40 → right of 40
6. **70**: 70 > 60 , 70 < 80 → left of 80
7. **90**: 90 > 60 , 90 > 80 → right of 80
8. **45**: 45 < 60 , 45 > 40 , 45 < 50 → left of 50
9. **85**: 85 > 60 , 85 > 80 , 85 < 90 → left of 90



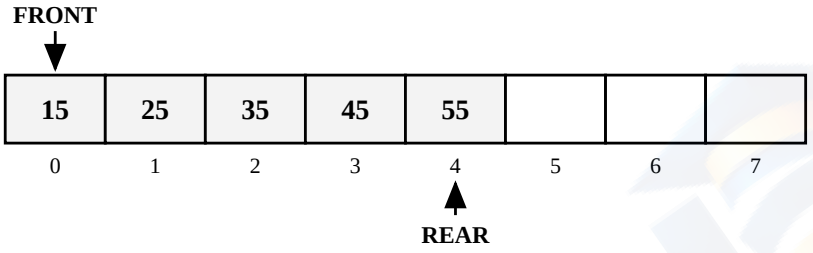
Final binary search tree

c) Effect of INSERT / DELETE on linear queue. 4 M

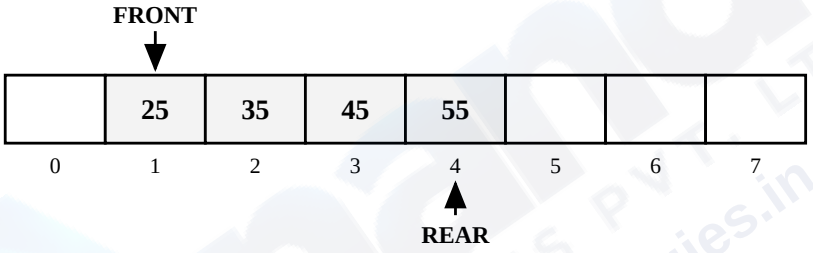
Size = 8 (index 0–7). Initially FRONT = 0, REAR = 3. INSERT increments REAR; DELETE increments FRONT.



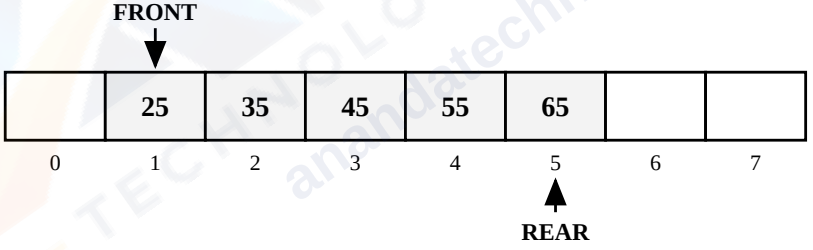
Initial: FRONT = 0, REAR = 3



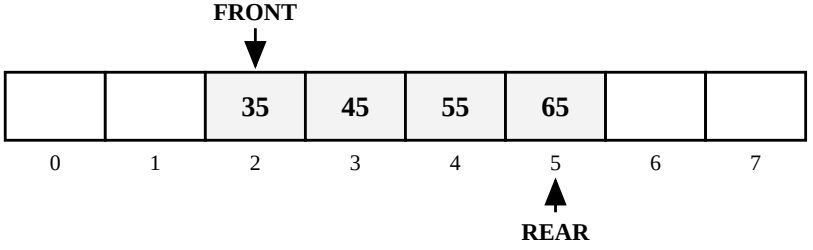
i) INSERT(55): REAR = 4



ii) DELETE: 15 removed, FRONT = 1



iii) INSERT(65): REAR = 5

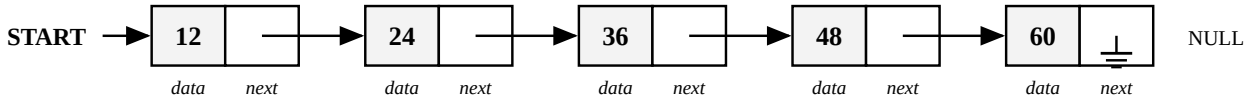


iv) DELETE: 25 removed, FRONT = 2

Final queue: 35, 45, 55, 65 (FRONT = 2, REAR = 5).

d) Create SLL 12, 24, 36, 48, 60 and search 48. 4 M

Creation: Each value is put in a new node and attached at the end; the last node's next = NULL.



Singly linked list after creation

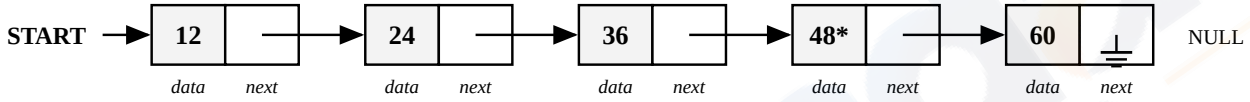
Search 48: Start with PTR = START and compare data of each node.

Step 1: PTR points to node 12. 12 = 48? No → PTR = PTR → next.

Step 2: PTR points to node 24. 24 = 48? No → PTR = PTR → next.

Step 3: PTR points to node 36. 36 = 48? No → PTR = PTR → next.

Step 4: PTR points to node 48. 48 = 48? Yes — node found at position 4.



** node 48 found (4th node)*

(Creation 2 M, search steps 2 M)

e) Stack is an ADT — justify. 4 M

An **Abstract Data Type** specifies the data and the operations on it, but hides how they are implemented.

- A stack is defined only by its behaviour (LIFO) and its operations: **PUSH(x)**, **POP()**, **PEEK()**, **isEmpty()**, **isFull()**.
- The user of a stack does not need to know whether it is stored in an **array** or a **linked list**; both give the same operations.
- Implementation details (TOP variable, array size, node pointers) are hidden from the user.

Hence, stack is an Abstract Data Type.

(Meaning of ADT 1 M, justification 3 M)

Q.5

a) C program — stack using array. 6 M

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

int stack[MAX];
int top = -1;

void push(int x)
{
```

```
    if (top == MAX - 1)
    {
        printf("Stack Overflow\n");
        return;
    }
    top = top + 1;
    stack[top] = x;
    printf("%d pushed\n", x);
}

void pop()
{
    if (top == -1)
    {
        printf("Stack Underflow\n");
        return;
    }
    printf("%d popped\n", stack[top]);
    top = top - 1;
}

void display()
{
    int i;
    if (top == -1)
    {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack (top to bottom): ");
    for (i = top; i >= 0; i--)
        printf("%d ", stack[i]);
    printf("\n");
}

int main()
{
    push(10);
    push(20);
    push(30);
    display();
    pop();
    display();
    return 0;
}
```

Output:

```
10 pushed
20 pushed
30 pushed
Stack (top to bottom): 30 20 10
30 popped
Stack (top to bottom): 20 10
```

b) Selection sort: {42, 17, 63, 8, 29, 51} 6 M

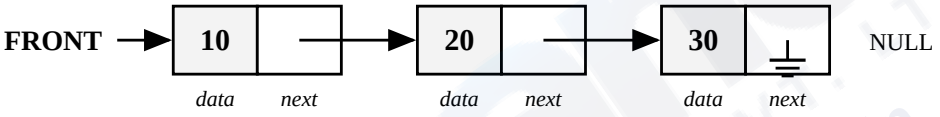
Working: In each pass the smallest element of the unsorted part is selected and swapped with the first element of the unsorted part.

Pass	Action	Array after pass
Initial	—	42, 17, 63, 8, 29, 51
1	smallest of remaining = 8 → position 0	8, 17, 63, 42, 29, 51
2	smallest of remaining = 17 → position 1	8, 17, 63, 42, 29, 51
3	smallest of remaining = 29 → position 2	8, 17, 29, 42, 63, 51
4	smallest of remaining = 42 → position 3	8, 17, 29, 42, 63, 51
5	smallest of remaining = 51 → position 4	8, 17, 29, 42, 51, 63

Sorted list: 8, 17, 29, 42, 51, 63

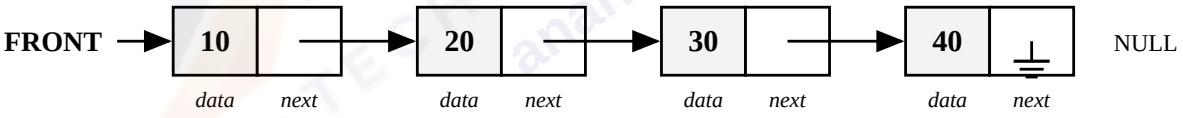
c) Queue using linked list. 6 M

In a linked queue each element is a node (data, next). Two pointers are kept: **FRONT** (first node, deletion end) and **REAR** (last node, insertion end). There is no overflow until memory is full.



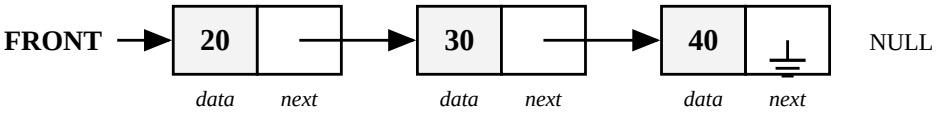
Linked queue: FRONT → first node, REAR → last node (30)

Insert (enqueue) x: create new node, new→data = x, new→next = NULL; if FRONT = NULL then FRONT = REAR = new, else REAR→next = new and REAR = new.



After insert 40 (REAR moves to 40)

Delete (dequeue): if FRONT = NULL → underflow; else temp = FRONT, FRONT = FRONT→next, free(temp) (if FRONT becomes NULL, REAR = NULL).



After delete (10 removed)

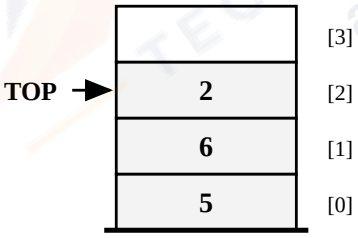
```
struct node { int data; struct node *next; } *front = NULL, *rear = NULL;
void insert(int x) {
    struct node *n = malloc(sizeof(struct node));
    n->data = x; n->next = NULL;
    if (rear == NULL) front = rear = n;
    else { rear->next = n; rear = n; }
}
void delete() {
    struct node *t;
    if (front == NULL) { printf("Underflow\n"); return; }
    t = front; front = front->next;
    if (front == NULL) rear = NULL;
    printf("%d deleted\n", t->data); free(t);
}
```

(Concept 2 M, insert 2 M, delete 2 M)

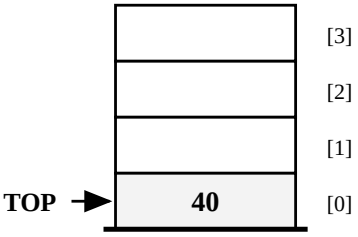
Q.6

a) Evaluate postfix: 5, 6, 2, +, *, 12, 4, /, − 6 M

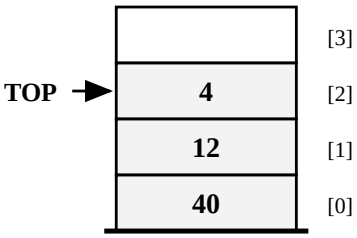
Symbol	Operation	Stack (bottom → top)
5	Push 5	5
6	Push 6	5, 6
2	Push 2	5, 6, 2
+	Pop 2, 6; 6 + 2 = 8; push 8	5, 8
*	Pop 8, 5; 5 * 8 = 40; push 40	40
12	Push 12	40, 12
4	Push 4	40, 12, 4
/	Pop 4, 12; 12 / 4 = 3; push 3	40, 3
-	Pop 3, 40; 40 - 3 = 37; push 37	37



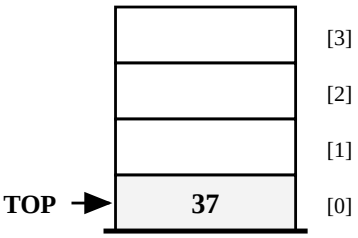
after 5, 6, 2



*after +, **



after 12, 4



final

Result = 37

b) Traversals of the given binary tree (Fig. 1). 6 M

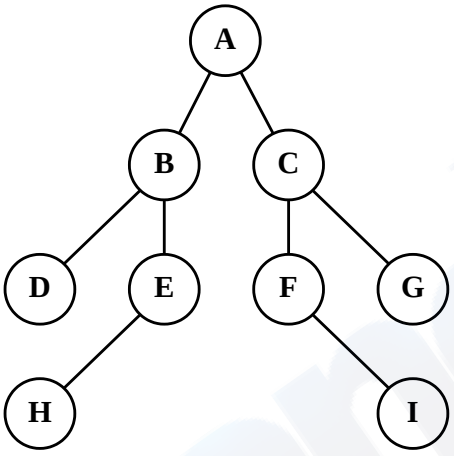


Fig. 1

Inorder (L, Root, R): D, B, H, E, A, F, I, C, G

Preorder (Root, L, R): A, B, D, E, H, C, F, I, G

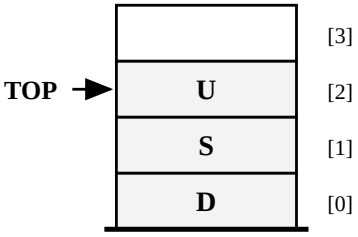
Postorder (L, R, Root): D, H, E, B, I, F, G, C, A

(Each traversal 2 M)

c) Two applications of stack with example. 6 M

1) Reversing a string: Push every character of the string on the stack, then pop all characters. Because of LIFO, they come out in reverse order.

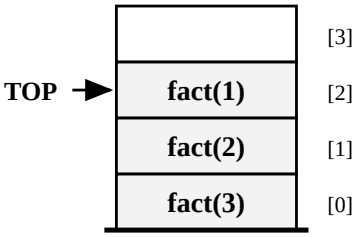
Example: "DSU" → PUSH D, S, U → POP gives U, S, D → "USD".



after pushing D, S, U

2) Recursion (function calls): Every function call is stored on the system stack with its local variables and return address; the last call returns first.

Example: fact(3) = 3 × fact(2) = 3 × 2 × fact(1). Calls are pushed as fact(3), fact(2), fact(1); fact(1) returns first, then fact(2) = 2, then fact(3) = 6.



call stack

(Other applications: infix to postfix conversion, postfix evaluation, balancing parentheses.)

(Each application with example 3 M)

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