(Affiliated to University of Mumbai)	***End Semester Examination (R-24/R-25) SH 2026 Answer Key with marking scheme***
Branch: **Computer Engg.**	Course: **Data Structures**
Year/ Semester: **SE / III**	Course code: **CEC302**
Time: 03 hours	Marks: 80

		Marks
Q. 1	All questions are compulsory.	30
A.	**Question:** What is an Abstract Data Type (ADT)? Differentiate between linear and non-linear data structures with two examples of each. **Solution:** **Abstract Data Type (1 mark)** An ADT is a mathematical model of a data type defined only by the values it can hold and the operations allowed on it, independent of how it is implemented. For example, a Stack ADT is defined by push, pop, peek and isEmpty; it may be implemented using an array or a linked list without changing how it is used. **Linear vs non-linear data structures (4 marks: any four points, 1 mark each)**	05
B.	**Question:** Using a stack, check whether the following expressions have well-formed parentheses. Show the stack contents after each bracket is read. (i) { [(a + b) * c] − (d / e) } (ii) [(a + b]) **Solution:** **Rule (1 mark)** Scan left to right. Push every opening bracket. On a closing bracket, the stack must not be empty and its top must be the matching opening bracket, which is then popped. The expression is well formed only if the stack is empty at the end. **(i) { [(a + b) * c] − (d / e) } (2 marks)**	05

Point	**Linear**	**Non-linear**
Arrangement	Elements are arranged in a sequence; each has at most one predecessor and one successor	Elements are arranged in a hierarchy or network; an element can be linked to many others
Traversal	All elements can be visited in a single run	Needs special traversals (e.g., inorder, DFS, BFS)
Levels	Single level	Multiple levels
Memory use	Simple to implement; may waste memory (arrays)	More complex; uses memory more flexibly
Examples	Array, stack, queue, linked list	Tree, graph

Symbol read	**Action**	**Stack (bottom → top)**
{	Push	{
[	Push	{ [
(	Push	{ [(
)	Top is (→ pop	{ [

]	Top is [→ pop	{
(	Push	{ (
)	Top is (→ pop	{
}	Top is { → pop	Empty

The stack is empty at the end, so expression (i) is well formed.

(ii) [(a + b]) (2 marks)

Symbol read	Action	Stack (bottom → top)
[	Push	[
(	Push	[(
]	Top is (but] needs [→ mismatch	[(

A mismatch occurs on reading], so expression (ii) is not well formed.

C. 05

Question:

Differentiate between an array and a linked list (any five points).

Solution:

Any five points, 1 mark each

Point	Array	Linked list
Memory allocation	Contiguous; size fixed at compile time (static)	Non-contiguous; nodes allocated at run time (dynamic)
Size	Fixed; may overflow or waste space	Grows and shrinks as needed
Access	Random access by index in O(1)	Sequential access from the head in O(n)
Insertion / deletion	Costly; elements must be shifted, O(n)	Easy once the position is known; only links change, O(1)
Extra memory	None per element	Extra pointer field(s) in every node
Cache performance	Good, due to locality	Poorer, since nodes are scattered

D. 05

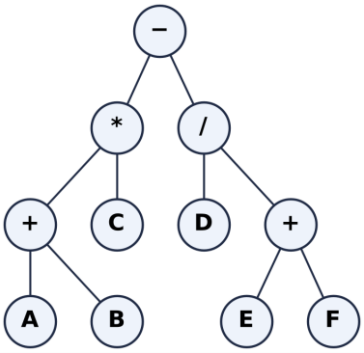
Question:

Construct the expression tree for (A + B) * C − D / (E + F). Write the preorder and postorder traversals of the tree.

Solution:

Expression tree (3 marks)

Operators with the lowest precedence, evaluated last, appear nearer the root. Here − is evaluated last, so it is the root; its left operand is (A + B) * C and its right operand is D / (E + F).



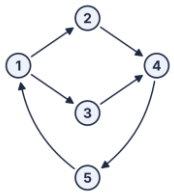
Traversals (2 marks)

Preorder (prefix): − * + A B C / D + E F

Postorder (postfix): A B + C * D E F + / −

E. **Question:** 05

Represent the following directed graph using (i) an adjacency matrix and (ii) an adjacency list.



Solution:

(i) Adjacency matrix (2.5 marks)

Entry [i][j] = 1 if there is an edge from vertex i to vertex j, otherwise 0.

	1	**2**	**3**	**4**	**5**
1	0	1	1	0	0
2	0	0	0	1	0
3	0	0	0	1	0
4	0	0	0	0	1
5	1	0	0	0	0

(ii) Adjacency list (2.5 marks)

```
1 → 2 → 3 → NULL
2 → 4 → NULL
3 → 4 → NULL
4 → 5 → NULL
5 → 1 → NULL
```

F. **Question:** 05

Insert the keys 72, 27, 36, 24, 63, 81, 92, 101 into a hash table of size 10 using the hash function h(k) = k mod 10. Resolve collisions using linear probing and show the final table.

Solution:

Calculation of home addresses and probing (3 marks)

Key	**h(k) = k mod 10**	**Probe sequence**	**Stored at**
72	2	2	2
27	7	7	7

36	6	6	6
24	4	4	4
63	3	3	3
81	1	1	1
92	2	2 (full), 3 (full), 4 (full), 5	5
101	1	1, 2, 3, 4, 5, 6, 7 (all full), 8	8

Final hash table (2 marks)

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Key	–	81	72	63	24	92	36	27	101	–

Linear probing uses h(k, i) = (h(k) + i) mod 10 for i = 0, 1, 2, … Key 92 needs 3 extra probes and key 101 needs 7, which shows the primary clustering caused by linear probing.

Q.2A

Question: 10

(i) Convert the infix expression A + (B * C − (D / E ^ F) * G) * H into postfix form using a stack. Show the stack and output after each symbol. (05)

(ii) Evaluate the postfix expression 5 6 2 + * 12 4 / − using a stack, showing every step. (05)

Solution:

(i) Infix to postfix (5 marks: rules 1, table 3, answer 1)

Operands go straight to the output. "(" is pushed. On ")", pop to the output until "(" is reached. For an operator, pop operators of higher precedence (or equal precedence, except for the right-associative ^) before pushing it. Precedence: ^ > * , / > + , −.

Symbol	**Stack**	**Postfix output**
A		A
+	+	A
(	+ (	A
B	+ (	A B
*	+ (*	A B
C	+ (*	A B C
−	+ (−	A B C *
(	+ (− (	A B C *
D	+ (− (	A B C * D
/	+ (− (/	A B C * D
E	+ (− (/	A B C * D E
^	+ (− (/ ^	A B C * D E
F	+ (− (/ ^	A B C * D E F
)	+ (−	A B C * D E F ^ /
*	+ (− *	A B C * D E F ^ /
G	+ (− *	A B C * D E F ^ / G
)	+	A B C * D E F ^ / G * −
*	+ *	A B C * D E F ^ / G * −
H	+ *	A B C * D E F ^ / G * − H
End		A B C * D E F ^ / G * − H * +

Postfix: A B C * D E F ^ / G * − H * +

(ii) Postfix evaluation (5 marks: method 1, steps 3, answer 1)

Push operands. On an operator, pop the top two values (first pop = right operand B, second pop = left operand A), compute A op B and push the result.

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

Result = 37

OR

Q.2B. 10

Question:

What is a circular queue and why is it preferred over a linear queue? Write algorithms for insertion and deletion in a circular queue implemented using an array. Show the queue (FRONT, REAR and contents) for a circular queue of size 5 after the operations: insert 10, 20, 30, 40; delete; delete; insert 50, 60, 70, 80.

Solution:

Definition and advantage (2 marks)

A circular queue is a queue in which the last position is connected back to the first, so that REAR moves to index 0 after index MAX − 1 using REAR = (REAR + 1) % MAX. In a linear array queue, positions freed by deletions at the front cannot be reused, so the queue can report overflow even when it has empty cells. A circular queue reuses those cells.

Insertion algorithm (2 marks)

```
ENQUEUE(Q, MAX, FRONT, REAR, ITEM)
1. If (REAR + 1) % MAX == FRONT: print "Overflow" and exit
2. If FRONT == -1: set FRONT = REAR = 0
   Else: set REAR = (REAR + 1) % MAX
3. Set Q[REAR] = ITEM
4. Exit
```

Deletion algorithm (2 marks)

```
DEQUEUE(Q, MAX, FRONT, REAR)
1. If FRONT == -1: print "Underflow" and exit
2. Set ITEM = Q[FRONT]
3. If FRONT == REAR: set FRONT = REAR = -1   (queue becomes
empty)
   Else: set FRONT = (FRONT + 1) % MAX
4. Return ITEM
```

Trace for MAX = 5 (4 marks)

Operation	FRONT	REAR	Q[0] Q[1] Q[2] Q[3] Q[4]
Initial	−1	−1	– – – – –
Insert 10	0	0	10 – – – –
Insert 20	0	1	10 20 – – –
Insert 30	0	2	10 20 30 – –
Insert 40	0	3	10 20 30 40 –

Delete (10)	1	3	– 20 30 40 –
Delete (20)	2	3	– – 30 40 –
Insert 50	2	4	– – 30 40 50
Insert 60	2	0	60 – 30 40 50
Insert 70	2	1	60 70 30 40 50
Insert 80	2	1	Overflow: (1 + 1) % 5 = 2 = FRONT

Insertions of 60 and 70 reuse cells 0 and 1, which a linear queue could not do. The queue is then full, so inserting 80 causes overflow.

Q.3A

10

Question:

Write algorithms (or C functions) for the following operations on a doubly linked list, and draw diagrams showing the links that change: (i) insert a node at the beginning, (ii) insert a node at the end, (iii) delete a node with a given value.

Solution:

Node structure (1 mark)

```
struct node {
    struct node *prev;
    int data;
    struct node *next;
};
struct node *start = NULL;
```

(i) Insert at the beginning (3 marks)

```
void insert_beg(int val) {
    struct node *nn = malloc(sizeof(struct node));
    nn->data = val; nn->prev = NULL; nn->next = start;
    if (start != NULL) start->prev = nn;
    start = nn;
}
```

Before: START → [10] ⇄ [20]. After inserting 5: START → [5] ⇄ [10] ⇄ [20]. Links changed: nn->next = old first node, old first->prev = nn, START = nn.

(ii) Insert at the end (3 marks)

```
void insert_end(int val) {
    struct node *nn = malloc(sizeof(struct node)), *p = start;
    nn->data = val; nn->next = NULL;
    if (start == NULL) { nn->prev = NULL; start = nn; return;
}
    while (p->next != NULL) p = p->next;   /* reach last node
*/
    p->next = nn; nn->prev = p;
}
```

Before: [10] ⇄ [20]. After inserting 30: [10] ⇄ [20] ⇄ [30]. Links changed: last->next = nn and nn->prev = last.

(iii) Delete a node with a given value (3 marks)

```
void delete_val(int val) {
    struct node *p = start;
    while (p != NULL && p->data != val) p = p->next;
    if (p == NULL) { printf("Not found"); return; }
    if (p->prev != NULL) p->prev->next = p->next;
```

```
    else start = p->next;                    /* deleting first
node */
    if (p->next != NULL) p->next->prev = p->prev;
    free(p);
}
```

Deleting 20 from [10] ⇄ [20] ⇄ [30]: 10->next becomes 30 and 30->prev becomes 10, giving [10] ⇄ [30].

Diagrams / explanation of changed links (1 mark)

OR

Q.3B

Question: 10

Explain how a queue is implemented using a singly linked list. Write C functions for insert (enqueue), delete (dequeue) and display. Show the linked list after the operations: insert 10, insert 20, insert 30, delete, insert 40.

Solution:

Concept (2 marks)

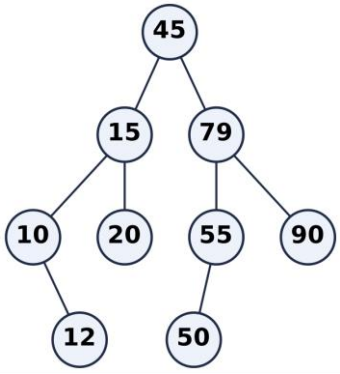
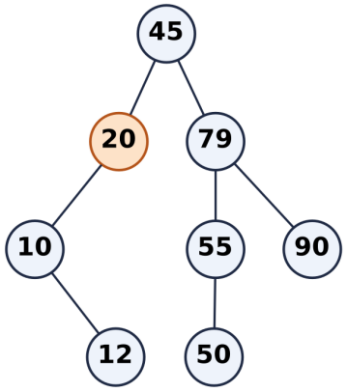
Two pointers are kept: FRONT points to the first node and REAR to the last. Insertion takes place at REAR and deletion at FRONT, so both operations take O(1) time. Since nodes are allocated dynamically, the queue never overflows unless memory is exhausted. The queue is empty when FRONT == NULL.

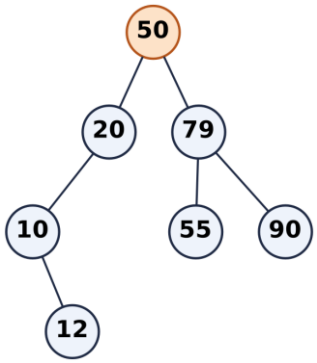
C functions (6 marks: enqueue 2, dequeue 2, display 2)

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

void enqueue(int val) {
    struct node *nn = malloc(sizeof(struct node));
    nn->data = val; nn->next = NULL;
    if (rear == NULL) front = rear = nn;      /* first node */
    else { rear->next = nn; rear = nn; }
}

int dequeue(void) {
    struct node *t; int val;
    if (front == NULL) { printf("Underflow"); return -1; }
    t = front; val = t->data; front = front->next;
    if (front == NULL) rear = NULL;          /* queue became
empty */
    free(t); return val;
}

void display(void) {
    struct node *p = front;
    while (p != NULL) { printf("%d ", p->data); p = p->next; }
}
```

Trace (2 marks)

Operation	Linked list (FRONT … REAR)
Insert 10	FRONT → [10] ← REAR
Insert 20	FRONT → [10] → [20] ← REAR
Insert 30	FRONT → [10] → [20] → [30] ← REAR
Delete	Returns 10; FRONT → [20] → [30] ← REAR
Insert 40	FRONT → [20] → [30] → [40] ← REAR

Q.4A	**Question:** Construct a Binary Search Tree by inserting the keys 45, 15, 79, 90, 10, 55, 12, 20, 50 in the given order. Write its inorder, preorder and postorder traversals. Then delete 15 followed by 45, and draw the tree after each deletion. **Solution:** **BST construction (3 marks)** Each key is compared with the root and moves left if smaller and right if larger, until an empty position is found.  **Traversals (3 marks)** Inorder: 10, 12, 15, 20, 45, 50, 55, 79, 90 (sorted order, which confirms the BST) Preorder: 45, 15, 10, 12, 20, 79, 55, 50, 90 Postorder: 12, 10, 20, 15, 50, 55, 90, 79, 45 **Delete 15 (2 marks)** Node 15 has two children. Replace it with its inorder successor, the smallest key in its right subtree, which is 20; then delete the original 20 (a leaf).  **Delete 45 (2 marks)** The root 45 has two children. Its inorder successor is 50, the leftmost node of the right subtree. Copy 50 into the root and delete the original 50 (a leaf).	10



OR

Q.4B

Question: 10

Construct the Huffman tree for the characters and frequencies given below. Find the Huffman code for each character and the total number of bits required to encode the message.

Character	**a**	**e**	**i**	**o**	**u**	**s**	**t**
Frequency	10	15	12	3	4	13	1

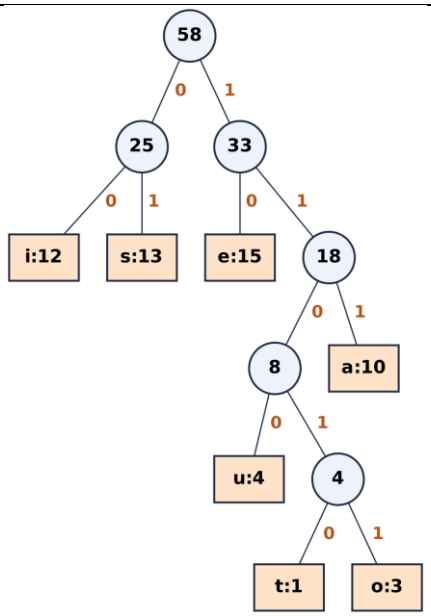
Solution:

Procedure (2 marks)

Repeatedly remove the two nodes with the smallest frequencies, make them children of a new node whose frequency is their sum (smaller on the left), and insert the new node back, until one node remains. Label left edges 0 and right edges 1.

Step	**Nodes merged**	**New node**
1	t (1) + o (3)	4
2	u (4) + [t,o] (4)	8
3	[u,t,o] (8) + a (10)	18
4	i (12) + s (13)	25
5	e (15) + 18	33
6	25 + 33	58 (root)

Huffman tree (4 marks)



Codes and total bits (4 marks)

Character	Frequency	Code	Code length	Bits
i	12	00	2	24
s	13	01	2	26
e	15	10	2	30
a	10	111	3	30
u	4	1100	4	16
t	1	11010	5	5
o	3	11011	5	15
Total	58			146

Total = 146 bits, compared with 58 × 3 = 174 bits using a fixed 3-bit code. Codes may differ if ties are broken differently, but the total stays 146 bits.

Q.5A

Question: 10

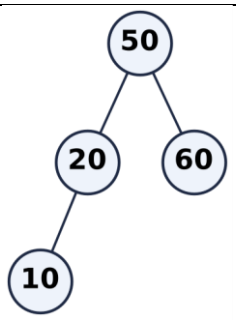
Construct an AVL tree by inserting the keys 50, 20, 60, 10, 8, 15, 32, 46, 11, 48 in the given order. Name each rotation performed and draw the tree after every rotation.

Solution:

Balance factor rule (1 mark)

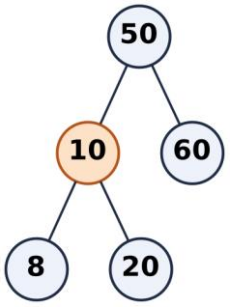
Balance factor (BF) = height of left subtree − height of right subtree. A node is balanced when BF is −1, 0 or +1. After each insertion, the nearest unbalanced ancestor is fixed with an LL, RR, LR or RL rotation.

Insert 50, 20, 60, 10 (1 mark) — no rotation



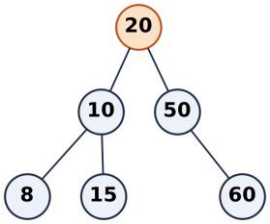
Insert 8 (2 marks) — LL rotation at 20

Node 20 gets BF = +2 and 8 was inserted in the left subtree of its left child, so a single right rotation at 20 makes 10 the parent of 8 and 20.

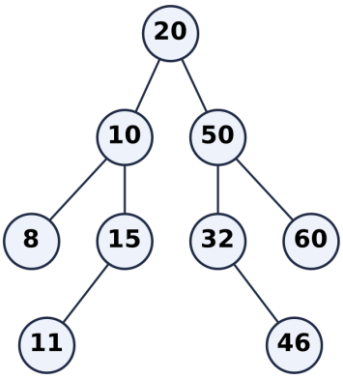


Insert 15 (2 marks) — LR rotation at 50

Node 50 gets BF = +2 and 15 was inserted in the right subtree of its left child (10 → 20 → 15). A left rotation at 10 followed by a right rotation at 50 makes 20 the new root.

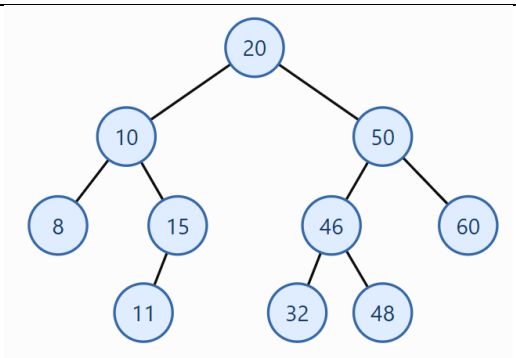


Insert 32, 46, 11 (1 mark) — no rotation



Insert 48 (2 marks) — RR rotation at 32

Node 32 gets BF = −2 and 48 was inserted in the right subtree of its right child (32 → 46 → 48), so a left rotation at 32 makes 46 the parent of 32 and 48.



Final AVL tree (1 mark)

Inorder of the final tree: 8, 10, 11, 15, 20, 32, 46, 48, 50, 60. Every node has BF in $\{-1, 0, +1\}$.

OR

Q.5B

Question: 10

(i) Construct a B-tree of order 3 by inserting the keys 10, 20, 30, 40, 50, 60, 70 in the given order. Show the tree after every split. (05)

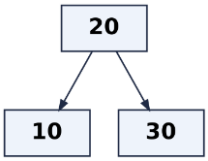
(ii) Differentiate between a B-tree and a B+ tree. (05)

Solution:

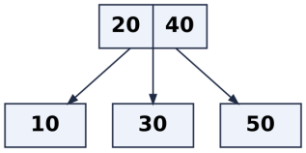
(i) B-tree of order 3 (5 marks)

In a B-tree of order 3 every node holds at most 2 keys and 3 children. When a node would hold 3 keys, it splits: the middle key moves up to the parent.

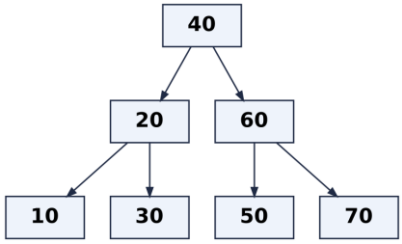
Insert 10, 20: [10 | 20]. Insert 30: [10 | 20 | 30] overflows, so 20 moves up.



Insert 40: [30 | 40]. Insert 50: [30 | 40 | 50] overflows, so 40 moves up to the root.



Insert 60: [50 | 60]. Insert 70: [50 | 60 | 70] splits and 60 moves up, making the root [20 | 40 | 60], which overflows; the root splits and 40 becomes the new root.



(ii) B-tree vs B+ tree (5 marks: any five points)

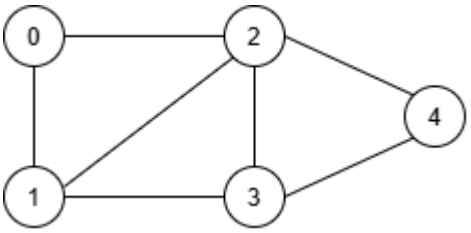
Point	B-tree	B+ tree

Data storage	Keys and records stored in internal and leaf nodes	Records stored only in leaf nodes; internal nodes hold keys for routing
Duplicate keys	Each key appears once	Internal keys are repeated in the leaves
Leaf linkage	Leaves are not linked	Leaves are linked as a list for sequential access
Search	May end at an internal node	Always ends at a leaf, so search time is uniform
Range queries	Slow; need tree traversal	Fast; follow the leaf links
Use	General indexing	Database and file-system indexes

Q.6A

Question: 10

Write the DFS and BFS traversals of the graph starting from vertex A. Show the contents of the stack at each step for DFS and the queue at each step for BFS.



Solution:

1. Depth First Search (DFS)

DFS traversal: `0 → 1 → 3 → 4 → 2`

Using an iterative stack, push adjacent unvisited vertices in descending order so the smallest is visited first.

Step	Vertex visited	Stack (bottom → top)
1	0	2, 1
2	1	2, 3
3	3	2, 4
4	4	2
5	2	Empty

2. Breadth First Search (BFS)

BFS traversal: `0 → 1 → 2 → 3 → 4`

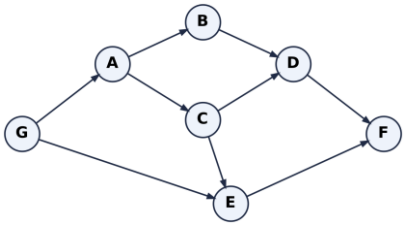
Step	Vertex visited	Queue (front → rear)
1	0	1, 2
2	1	2, 3
3	2	3, 4
4	3	4
5	4	Empty

OR

Q.6B

Question: 10

Explain topological sort. Find a topological ordering of the directed acyclic graph given below using the in-degree (Kahn's) method. Show the in-degree of every vertex at each step.



Solution:

Explain **Topological sort- 4 marks**

Topological sorting is a linear ordering of the vertices of a directed acyclic graph (DAG) such that, for every directed edge u -> v, vertex u appears before vertex v.

Kahn's algorithm uses the in-degree of each vertex, where the in-degree is the number of incoming edges.

Steps:

1. Calculate the in-degree of every vertex.
2. Add all vertices with in-degree 0 to a queue.
3. Remove a vertex from the queue and add it to the topological ordering.
4. Decrease the in-degree of each of its outgoing neighbours by 1.
5. Add any neighbour whose in-degree becomes 0 to the queue.
6. Repeat until the queue is empty.

(i) Topological sort (6 marks)

Repeatedly output a vertex with in-degree 0 and reduce the in-degree of its successors. When several vertices have in-degree 0, the alphabetically smallest is taken.

Step	Output	In-degree A B C D E F G	Vertices with in-degree 0
Initial	–	1 1 1 2 2 2 0	G
1	G	0 1 1 2 1 2 –	A
2	A	– 0 0 2 1 2 –	B, C
3	B	– – 0 1 1 2 –	C
4	C	– – – 0 0 2 –	D, E
5	D	– – – – 0 1 –	E
6	E	– – – – – 0 –	F
7	F	– – – – – – –	–

Topological order: G, A, B, C, D, E, F

	Other valid orders exist, e.g. G, A, C, B, E, D, F; any order in which every edge u → v has u before v is correct.	
********** All the Best**********		