USN | | | | | | | | | | |

Internal Assessment Test 1

Sub:	DATA STRUCTURES AND APPLICATIONS					Sub Code:	BCS304	Branch:	AIML,CSE-AIML	
Date:		Duration:	90 minutes	Max Marks:	50	Sem/Sec:	III -A, B, C		OBE	

		Answer any FIVE FULL Questions	MARKS	CO	RBT
1	a	Write the code snippets for isFull() and isEmpty() of a Stack.Consider a stack where the data are pushed to (PUSH operation) and popped from (POP operation) in the following order: PUSH 3; TOP; PUSH 7; TOP; PUSH 6; PUSH 9; TOP; POP; POP; TOP; where the PUSH inserts an item onto the stack, POP deletes an item from the stack and TOP returns the top position element of the stack. Write the values returned by TOP for the sequence of operations above.	4 M	1	L3

	b	Differentiate between Structure and union with a suitable example. Write the output for the below code snippet.
		
		#include<*stdio.h*>
		
		struct st
		{
		int x;
		struct st *next;
		};
		
		int main()
		{
		struct st temp;
		temp.x = 10;
		temp.next = temp;
		printf("%d", temp.next.x);
		return 0;
		}
			6M	1	L3

2	a	Write and apply the algorithm to convert the below infix expression to a postfix expression (((a/b)-c)+(d*e))-(a*c)	5M	1	L3
	b	Write an algorithm to evaluate a postfix expression. Trace the algorithm for the following expression showing the stack contents 6 5 1-423^/+.	5M	1	L3
3	a	Develop functions to implement the following using singly linked list: i)Delete a node from the front ii) Concatenate two linked lists.	6M	2	L2
	b	Explain the representation of a sparse matrix (using Linked List and Array). 0 0 3 0 4 0 0 5 7 0 0 0 0 0 0 0 2 6 0 0	4M	1	L2
4	a	What is Circular Queue in a Data Structure?Explain the operations.	4M	2	L2
	b	Develop C functions to implement the following in a doubly linked list: i) Insert a node at the front ii) Delete a node from the end.	6M	3	L2
5		What is a Doubly Linked List? Explain the representation of Doubly Linked List .Write the code for the following: a) print all the data of nodes . b) Find the length of list.	10M	3	L2
6	a	Write a program to solve tower of Hanoi problem. Trace it for 3 disks with diagram.	5 M	1	L3
	b	Write the C function to add two polynomials. Show the linked representation of the below two polynomials and their addition using a circular singly linked list P1: 5x3 + 4x2 +7x + 3 P2: 6x2 + 5 Output: add the above two polynomials and represent them using the linked list.	5M	2	L3

CI **CCI** **HOD**

----------All the Best----------

1 a)

isFull() (1M)

```
{
    If(top==N-1)
    Printf("Stack OverFlow");
}
```

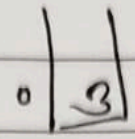
isEmpty() (1M)

```
{
    If(top==-1)
    Printf("stack is empty");
}
```

---

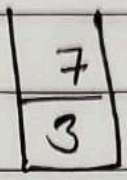
2 marks

PUSH 3



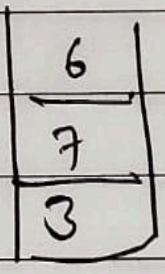
TOP element = 3

PUSH 7

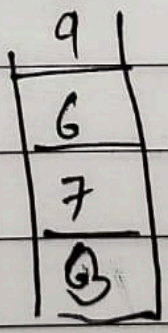


TOP element = 7

PUSH 6

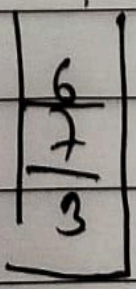


PUSH 9

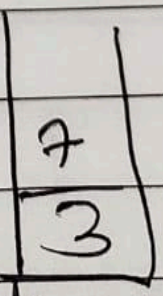


TOP element = 9

POP



POP



TOP ele = 7

b) ----------2 M(any 2 with example)

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

b) ERROR----------4 M

2 a)Algorithm----------2M

1. Scan the infix expression **from left to right**.
2. If the scanned character is an operand, put it in the postfix expression.
3. Otherwise, do the following
    - If the precedence of the current scanned operator is higher than the precedence of the operator on top of the stack, or if the stack is empty, or if the stack contains a '(', then push the current operator onto the stack.
    - Else, pop all operators from the stack that have precedence higher than or equal to that of the current operator. After that push the current operator onto the stack.
4. If the scanned character is a '**(**', push it to the stack.
5. If the scanned character is a '**)**', pop the stack and output it until a '**(**' is encountered, and discard both the parenthesis.
6. Repeat steps **2-5** until the infix expression is scanned.

7. Once the scanning is over, Pop the stack and add the operators in the postfix expression until it is not empty.
8. Finally, print the postfix expression.

ANSWER

ab/c-de*+ac*-

Infix to Postfix table----------3M

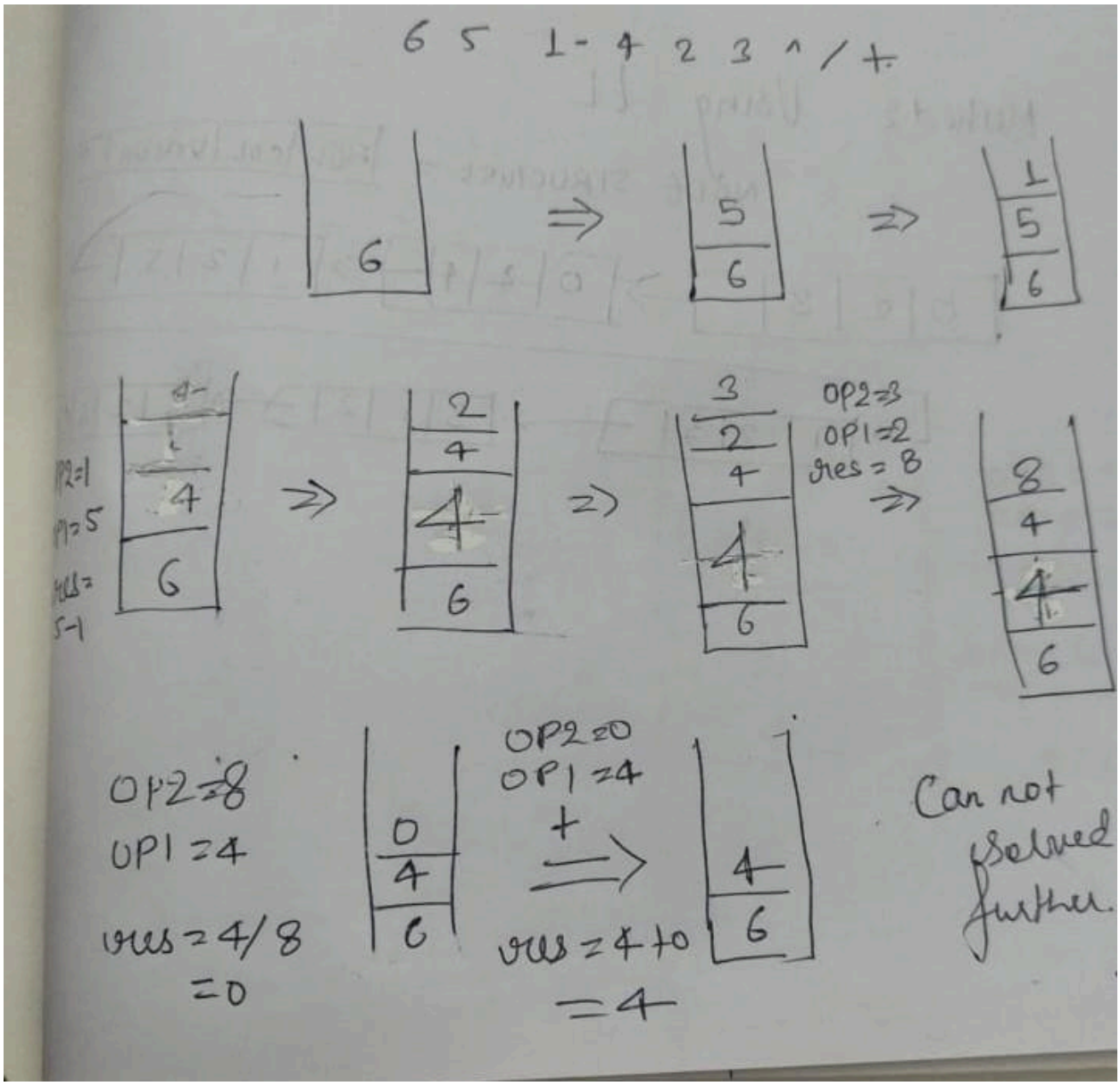
Sr No	Expression	Stack	Postfix
0	(	(	
1	(	((	
2	(	(((	
3	a	(((	a
4	/	(((/	a
5	b	(((/	a b
6	)	((	a b /
7	-	((-	a b /
8	c	((-	a b / c
9	)	(	a b / c -
10	+	(+	a b / c -
			l
11	(	(+ (	a b / c -
12	d	(+ (	a b / c - d
13	*	(+ (*	a b / c - d
14	e	(+ (*	a b / c - d e
15	)	(+	a b / c - d e *
16	)		a b / c - d e * +

Sr No	Expression	Stack	Postfix
17	-	-	a b / c - d e * +
18	(	- (	a b / c - d e * +
19	a	- (	a b / c - d e * + a
20	*	- (*	a b / c - d e * + a
21	c	- (*	a b / c - d e * + a c
22	)	-	a b / c - d e * + a c *
23			a b / c - d e * + a c * -

2 b)----------2M

- Create a stack to store operands (or values).
- Scan the given expression from left to right and do the following for every scanned element.
    - If the element is a number, push it into the stack.
    - If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack.
- When the expression is ended, the number in the stack is the final answer.

  ----------3 M

- 

3a. i)----------3M

```
struct Node*   deleteHead(struct Node*  head) {

   if (head == NULL)
      return NULL;
   NODE  temp = head;
```

```
    head = head->next;
    free(temp);
return head;
}
```

ii)----------3M

```
struct Node *concat(struct Node *head1,  struct Node *head2) {
  if (head1 == NULL)
        return head2;

    struct Node *curr = head1;
    while (curr->next != NULL){
        curr = curr->next;
    }
    curr->next = head2;
    return head1;
}
```

b)----------2M

A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have **0 value**, then it is called a sparse matrix.

Why to use Sparse Matrix instead of simple matrix ?

- **Storage:** There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
- **Computing time:** Computing time can be saved by logically designing a data structure traversing only non-zero elements..
- **Example:**

- ```
  0 0 3 0 4
  0 0 5 7 0
  0 0 0 0 0
  0 2 6 0 0
  ```
- Representing a sparse matrix by a 2D array leads to wastage of lots of memory as zeroes in the matrix are of no use in most of the cases. So, instead of storing zeroes with non-zero elements, we only store non-zero elements. This means storing non-zero elements with **triples-(Row, Column, value).**
- Sparse Matrix Representations can be done in many ways following are two common representations:
  Array representation
  Linked list representation
  ----------2 M

Method 1: Using Arrays:

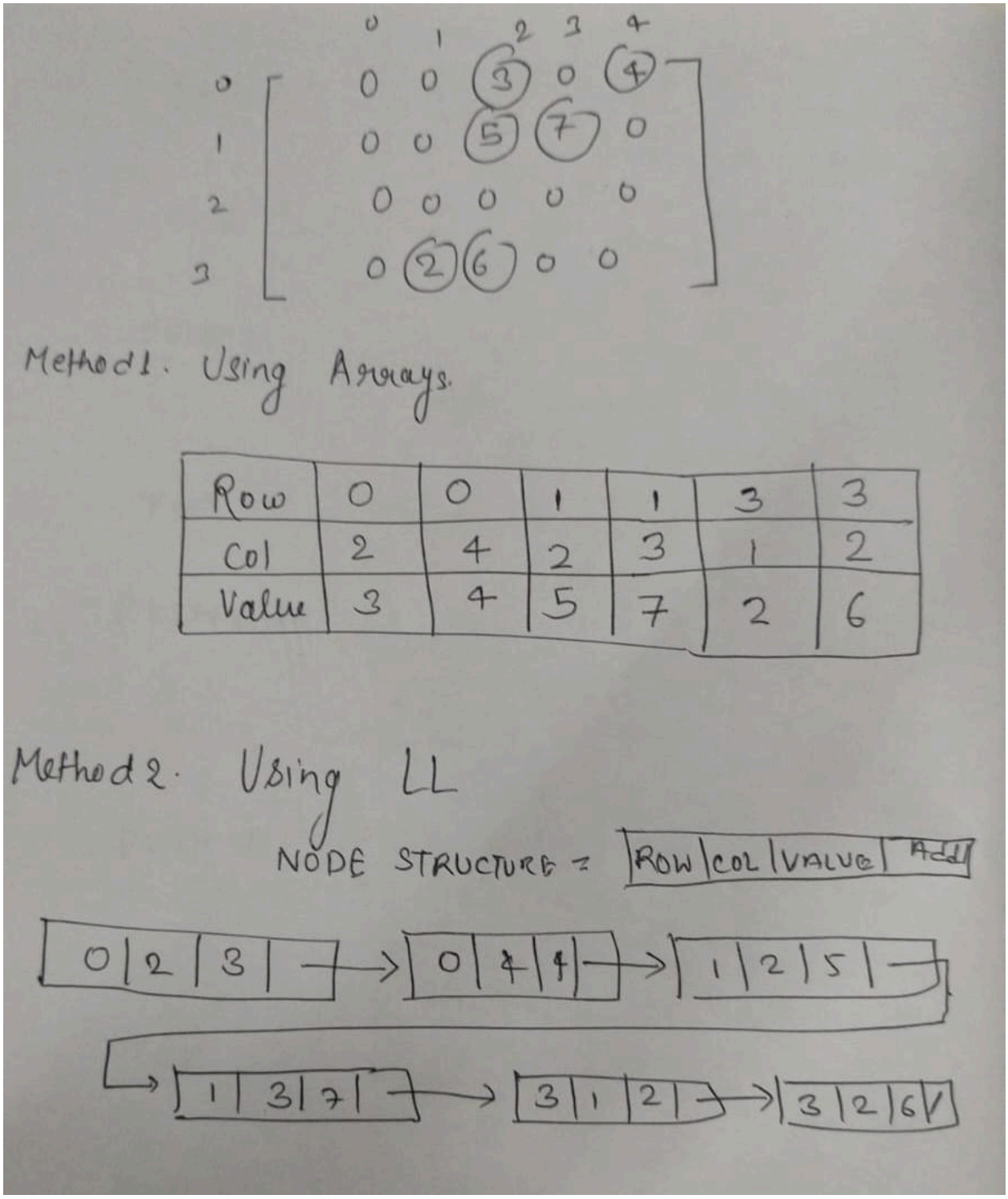
2D array is used to represent a sparse matrix in which there are three rows named as

- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row,column)

Method 2: Using Linked Lists
In linked list, each node has four fields. These four fields are defined as:

- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row,column).

- **Next node:** Address of the next node
  ```

- 
0 1 2 3 4
0 0 0 3 0 4
1 0 0 5 7 0
2 0 0 0 0 0
3 0 2 6 0 0
Method 1. Using Arrays.
Row 0 0 1 1 3 3
Col 2 4 2 3 1 2
Value 3 4 5 7 2 6
Method 2. Using LL
NODE STRUCTURE = ROW COL VALUE Addr
0 2 3
0 4 4
1 2 5
1 3 7
3 1 2
3 2 6

4 a)

Circular Queue is an extended version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle. In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue.---------2 M

- **enQueue(value)** This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at the rear position. 1 M
    - Check whether the queue is full – [i.e., the rear end is in just before the front end in a circular manner].
    - If it is full then display Queue is full.
        - If the queue is not full then, insert an element at the end of the queue.
- **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from the front position. 1 M
    - Check whether the queue is Empty.
    - If it is empty then display Queue is empty.
        - If the queue is not empty, then get the last element and remove it from the queue.

4b)----------3 M

i)
```
struct Node *insertAtFront(struct Node *head, int new_data) {
   struct Node *new_node = createNode(new_data);
   new_node->next = head;
   if (head != NULL) {
      head->prev = new_node;
   }
   return new_node;
}
```

ii)

----------3 M

```
struct Node* delLast(struct Node *head) {


   if (head == NULL)
      return NULL;
   if (head->next == NULL) {
      free(head);
      return NULL;
   }
   struct Node *curr = head;
   while (curr->next != NULL)
   curr->prev->next = NULL;
   free(curr);
   return head;
}
```

5.

A **doubly linked list** is a data structure that consists of a set of nodes, each of which contains a **value** and **two pointers**, one pointing to the **previous node** in the list and one pointing to the **next node** in the list. This allows for efficient traversal of the list in **both directions**, making it suitable for applications where frequent **insertions** and **deletions** are required.-2M

a) void printList(Node* head)----------4M

```
{
   Node* temp = head;
   printf("Forward List: ");
   while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
```

```
    }
    printf("\n");
}
```

B)----------4 M

```
int findSize(struct Node* curr) {
  int size = 0;
  while (curr != NULL) {
    size++;
    curr = curr->next;
  }
  return size;
}
```

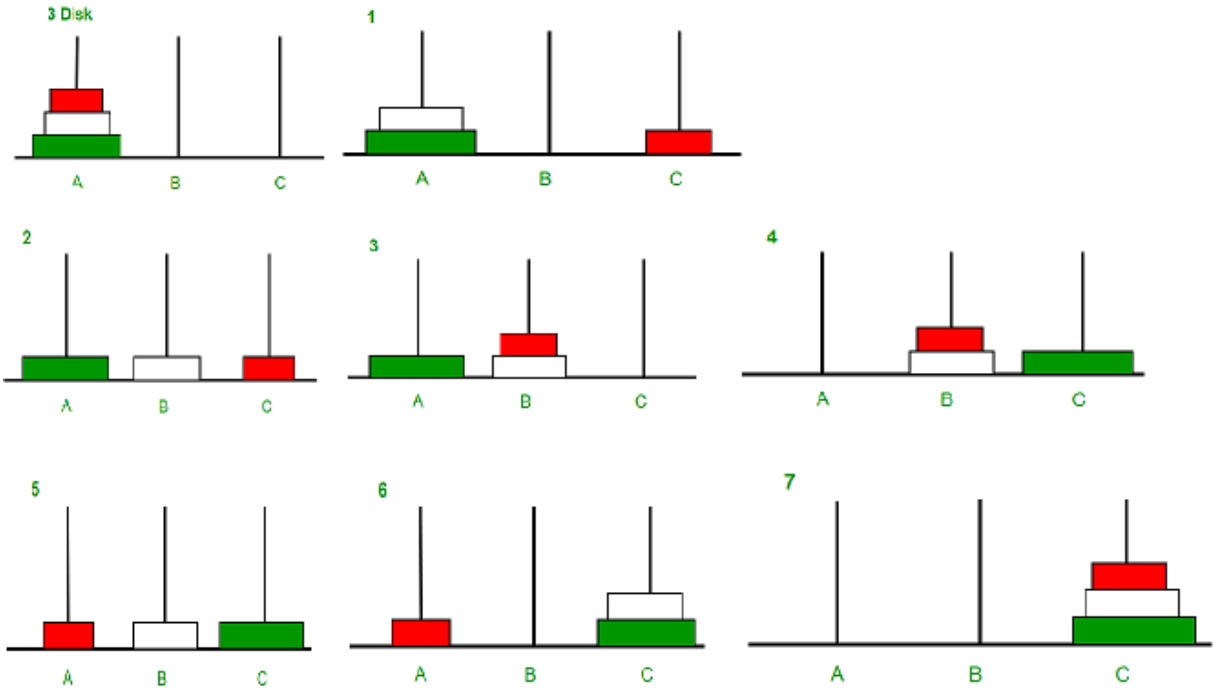
6a)----------2M----------

```
  void towerOfHanoi(int n, char from_rod, char  to_rod,char  aux_rod)
{
    if (n == 0) {
        return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    cout << "Move disk " << n << " from rod " << from_rod << " to rod " << to_rod << endl;
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}

// Driver code
int main()
{
```

```
    int N = 3;
    towerOfHanoi(N, 'A', 'C', 'B');
    return 0;
}
```

----------3M----------



Move disk 1 from rod A to rod C

Move disk 2 from rod A to rod B

Move disk 1 from rod C to rod B

Move disk 3 from rod A to rod C

Move disk 1 from rod B to rod A

Move disk 2 from rod B to rod C

Move disk 1 from rod A to rod C

6a)----------2M

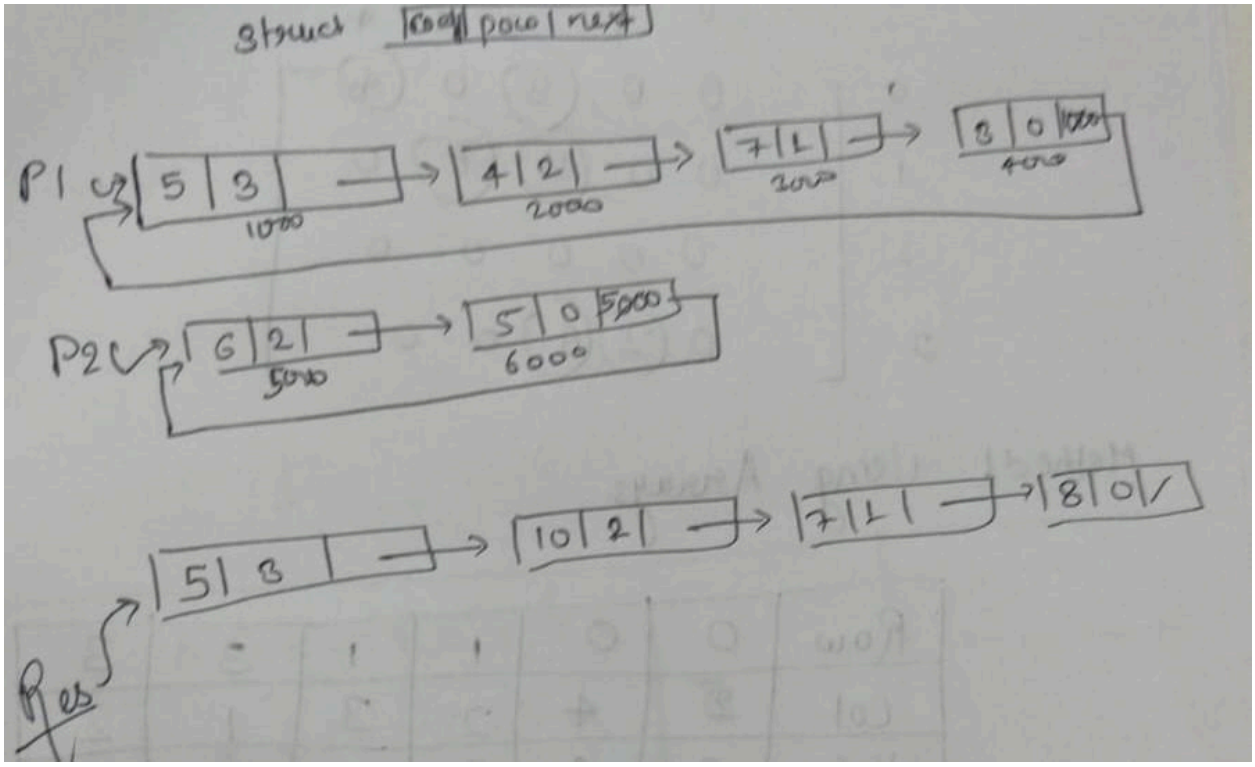
```
struct Node* addPolynomial(struct Node* head1, struct Node* head2) {
```

```
    if (head1 == NULL) return head2;
    if (head2 == NULL) return head1;

    if (head1->pow > head2->pow) {
        struct Node* nextPtr =addPolynomial(head1->next, head2);
        head1->next = nextPtr;
        return head1;
    }
    else if (head1->pow < head2->pow) {
        struct Node* nextPtr = addPolynomial(head1, head2->next);
        head2->next = nextPtr;
        return head2;
    }

    struct Node* nextPtr =  addPolynomial(head1->next, head2->next);
    head1->coeff += head2->coeff;
    head1->next = nextPtr;
    return head1;
}
```

---------3 M


struct
coef
pow
next
P1
5
3
1000
4
2
2000
7
1
3000
8
0
4000
P2
6
2
5000
5
0
6000
Res
5
3
10
2
7
1
8
0