PAGE NO.:
DATE: / /

DS OBA - 2

2GI19CS175
Venkatesh G D

1.)
```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
void push(struct Node** head_ref, int new_data);
bool isPresent(struct Node* head, int data);

struct Node* getUnion(struct Node* head1, struct Node* head2){
    struct Node* result = NULL;
    struct Node *t1 = head, *t2 = head2;
    while (t1 != NULL) {
        push(&result, t1->data);
        t1 = t1->next;
}

struct Node* getIntersection(struct Node* head1, struct Node* head2){
  struct Node* result = NULL;
  struct Node* t1 = head;
  while(t1 != NULL){
      if (isPresent(head2, t1->data))
          push(&result, t1->data);
      t1 = t1->next;
  }
  return result;
}
```

PAGE NO.:
DATE : / /

1.)
```
void push (struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(
                                sizeof (struct Node));
    new_node -> data = new_data;
    new_node -> next = (*head_ref);
    (*head_ref)      = new_node;
}

void printList (struct Node* node) {
    while (node != NULL) {
        printf ("%d", node -> data);
        node = node -> next;
    }
}

bool isPresent (struct Node* head, int data) {
    struct Node* t = head;
    while (t != NULL) {
        if (t -> data == data)
            return 1;
        t = t -> next;
    }
    return 0;
}

int main()
{
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
    struct Node* intersection = NULL;
    struct Node* union = NULL;
```

1)

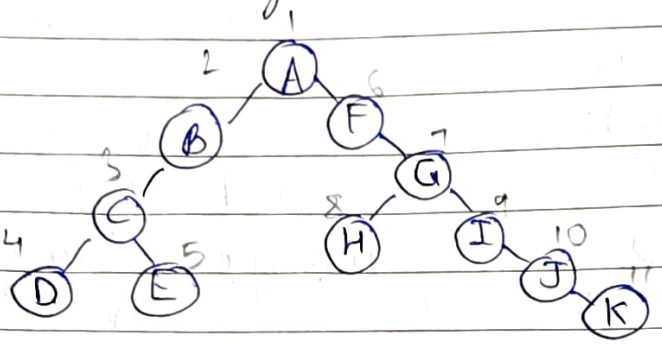
```
push(&head1, 23);
push(&head1, 14);
push(&head1, 18);
push(&head1, 31);
push(&head1, 9);

push(&head2, 12);
push(&head2, 23);
push(&head2, 9);
push(&head2, 5);
push(&head2, 25);

intersection = getIntersection(head1, head2);
union = getUnion(head1, head2);

printf("\n First List :\n");
printList(head1);

printf("\n Second list : \n");
printList(head2);

printf("\n Intersection :");
printList(intersection);

printf("\n Union :");
printList(union);

return 0;

}
```

// Executed & tested on VS code, Ubuntu 20.8.1.

2) i) Pre-Order Traversal

→

1. Visit Root
2. Traverse the left subtree, i.e call preorder (left subtree)
3. Traverse right subtree, i.e call preorder (right subtree)



	Recursive call	Output
1.	Visit Root	A
2.	Recur left child of Root A	A B
3.	Recur left child of B	A B C
4.	Recur left child of C.	A B C D
5.	Recur right child (Because left is NULL)	A B C D E
6.	Recur right child of Root A	A B C D E F
7.	Recur right child of F (Because left child of F is null)	A B C D E F G
8.	Recur left child of G	A B C D E F G H
9.	Recur right child of G	A B C D E F G H I
10.	Recur right child of I (Because left is NULL)	A B C D E F G H I J
11.	Recur right child of J (Because left is NULL)	A B C D E F G H I J K

∴ Final Output : A B C D E F G H I J K
(PREORDER)

2.) ii.) Inorder Traversal

→

1. Visit left sub-tree, i.e call Inorder (left-subtree)
2. Visit Root
3. Traverse right subtree, i.e call Inorder (right-subtree).

Recursive Calls	Output
1.) Recur on left subtree (Traverse A → B → C → D)	D
2.) Visit Root (D → C)	DC
3.) Recur on right subtree (C → E)	DCE
4.) Visit Root (C → B)	DCEB
5.) Vist Root, since right subtree of B is NULL (B → A)	DCEBA
6.) Recur right subtree of A. (A → F)	DCEBAF
7.) Recur to right subtree & then recur left subtree (F → G → H)	DCEBAFH
8.) Visit Root of H (H → G)	DCEBAFHG
9.) Recur to right subtree (G → I)	DCEBAFHGI
10.) Recur to right subtree (I → J)	DCEBAFHGIJ
11.) Recur to right subtree (J → K)	DCEBAFHGIJK

Final Output : DCEBAFHGIJK
(INORDER)

2.) iii.) Post Order Traversal.

1. Traverse left subtree, i.e call Postorder (left-subtree)
2. Traverse right subtree, i.e call Postorder (right-subtree)
3. Visit the root.

Recursion calls	Output
1. Recur on left subtree (A→B→C→D)	D
2. Traverse right subtree (D→E)	DE
3. Visit Root (E→C)	DEC
4. Visit Root (C→B)	DECB
5. Traverse right subtree & recur to left subtree (B→F→G→H)	DECBH
6. ~~Visit Root (H→K)~~	~~DECBHK~~
6. Recur to right subtree (H→I→J→K)	DECBHK
7. Visit Root (K→J)	DECBHKJ
8. Visit Root (J→I)	DECBHKJI
9. Visit Root (I→G)	DECBHKJIG
10. Vist Root (G→F)	DECBHKJIGF
11. Visit Root (F→A)	DECBHKJIGFA

Final Output : DECBHKJIGFA
(POSTORDER)