Q.) Delete the middle node of a linked list. → Leet code.

Ans →

```
struct ListNode * deleteMiddle (struct ListNode* head) {
    int n = 0;
    struct ListNode * current = head,
      * nextnode,   * prevnode;
    while ( current != NULL) {
          current = current → next;
          n++;
    }
   int mid = n/2;        int i=0;
   current = head;
   if (mid == 0)
  {
          free (current)
          head = NULL

           return head;
    }
 else if (mid==1)
 {
          prevnode =     current;
           current =     current → next;
            prevnode → next = current → next
               free ( current);
               return head;
 }
else {
       while( i< mid-1)
   {
            current = current → next;
             nextnode = current → next;
             i++
    current -> next = nextnode -> next;
    free ( nextnode ) ;
    return head;
}
```

Q.) Odd even linked list -> leetcode

Ans->

```
struct ListNode* OddEvenList ( struct ListNode
                                        *head ) {
    if ( !head || !head->next ||
                  !head-> next -> next ) {
        return head;
    }

    struct ListNode* oddHead = head, *evenHead
           = head-> next;

    struct ListNode *odd = oddHead, *even= evenHead;
    while( even && even-> next ) {
        odd -> next = even-> next;
        odd = odd -> next;
        even-> next = odd -> next;
        even = even-> next;
    }
    odd -> next = evenHead;
    return oddHead;
}
```

Q.) Write a program

a.) to construct Binary search tree

b.) Traverse the tree using inorder, postorder, preorder.

c.) Display the elements in tree.

Ans ⇒

```
# include < stdio.h >
# include <  stdlib.h >
struct  TreeNode {
        int val;
        struct TreeNode * left;
        struct TreeNode * right;
};

struct TreeNode * createNode ( int val) {
    struct TreeNode * newNode
        = ( struct TreeNode * ) malloc
        ( sizeof (struct  TreeNode ));
        newnode → val = val;
        newNode → left = NULL;
        newNode → right = NULL;
        return newNode;
}
struct  TreeNode * insert ( struct
    TreeNode * root , int val ) {
        if ( root == NULL)
    {   return  createNode ( val );
    }
    if ( val < root → val ) {
            root → left = insert ( root → left,
                                    val );
    }
else if ( val > root -> val ) {
        root -> right = insert ( root -> right,
                                          val );
   }
   return root;
}

void inorderTraversal ( struct TreeNode *root)
{
      if ( root == NULL ) {
            return;
      }
   inorderTraversal ( root -> left );
   printf ( " %d", root -> val );
   inorderTraversal ( root -> right );
}

void preorderTraversal ( struct TreeNode* root){
      if ( root == NULL ) {
               return ;
   }

printf( "%d" , root -> val );
   preorderTraversal ( root -> left );
   preorderTraversal ( root -> right);
}
void postorderTraversal( struct TreeNode*
                                    root) {

      if ( root == NULL ) {
               return;
      }

 postorderTraversal ( root -> left );
 postorderTraversal ( root -> right);
  printf ( " %d" , root -> val );
}
```

```
void  displayTree (struct TreeNode* root) {
     printf ("Elements  in  tree : ");
    inorderTraversal (root);
      printf ("\n");
}
int main()
{
     struct TreeNode* root = NULL;
       int  choice = -1, val;
     printf ("\n1. Insert element \n");
      printf ("2. Display tree (inorder)\n");
     printf (" 3. Display tree (preorder)
                                        \n");
    printf ("4. Display tree (postorder)\n");
     printf ("5. Exit \n");

while (choice != 5) {
     printf ("Enter your choice: ");
       scanf ("%d", &choice);
     switch (choice) {
           case 1:
              printf ("Enter value to insert: ");
              scanf ("%d", &val);
         root =  insert(root, val);
         break;

   Case 2:
          displayTree (root);
           break;

 case 3:
        printf ("elements in tree
                         (preorder): ");
    preorderTraversal (root);
    printf ("\n");
    break;

  case 4:
      printf ("Elements in tree (postorder):
                                            ");
    postorderTraversal (root);
    break;

  case 5:
        return 0;
  }
 }
```

Output -:

```
Enter your choice: 1
Enter value to insert : 33
Enter your choice : 1
Enter value to insert : 44
Enter choice : 2
Elements in tree : 33 44
Enter choice : 3
Elements in tree (preorder) : 33 44
Enter choice : 5
Exit
```

19.02.24