

LAB ASSIGNMENT

- 1) Write a C program to print pre order, inorder and postorder transversal on binary tree.

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

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

```
struct node
```

```
{
```

```
    int data;
```

```
    int value;
```

```
    struct node * left;
```

```
    struct node * right;
```

```
}
```

```
void inorder (struct node *root)
```

```
{
```

```
    if (root == NULL) return
```

```
        return;
```

```
    inorder (root->left);
```

```
        printf ("%d->", root->data);
```

```
    inorder (root->right);
```

```
}
```

```
void pre order (struct node *root)
```

```
{
```

```
    if (root == NULL)
```

```
        return;
```

```
    printf ("%d->", root->data);
```

```
    pre order (root->left);
```

```
        pre order (root->right);
```

```
}
```

```

void post_order(struct node *root)
{
    if (root == NULL)
        return;
    post_order(root->left);
    post_order(root->right);
    printf("%d->", root->data);
}

struct node *create_node(value)
{
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}

```

```

void main()
{
    struct node *root = create_node(1);
    root->left = create_node(12);
    root->right = create_node(9);
    root->left->left = create_node(10);
    root->left->right = create_node(15);
    root->right->left = create_node(11);
    root->right->right = create_node(16);

    printf("In order traversal\n");
    in_order(root);
    printf("\n pre order traversal\n");
    pre_order(root);
}

```

```
printf("In postorder traversal\n");  
postorder(root);
```

```
}
```

Output:

Inorder traversal

10 → 12 → 15 → 1 → 11 → 9 → 16 →

pre order traversal

1 → 12 → 10 → 15 → 9 → 11 → 16 →

postorder traversal

10 → 15 → 12 → 11 → 16 → 9 → 1 →

Q.) write a c program to create (or insert) and inorder traversal on binary search tree.

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

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

```
struct Node
```

```
{
```

```
int key;
```

```
struct node * left, * right;
```

```
}
```

```
struct node * new Node(int item)
```

```
{
```

```
struct node * temp = (struct node *) malloc(sizeof(struct node));
```

```
temp->key = item;
```

```
temp->left = temp->right = NULL
```

```
return temp;
```

```
}
```

```
void inorder(struct node * root)
```

```

{
    if (root != NULL)
    {
        inorder (root->left);
        printf ("%d\n", root->key);
        inorder (root->right);
    }
}

```

```

}

struct node* insert (struct node *node, int key)
{
    if (node == NULL) return new Node (key);
    if (key < node->key)
        node->left = insert (node->left, key);
    else if (key > node->key)
        node->right = insert (node->right, key);
    return node;
}

```

```

}

int main ()
{
    struct node * root = NULL;
    root = insert (root, 3);
    insert (root, 12);
    insert (root, 51);
    insert (root, 43);
    insert (root, 37);
    insert (root, 98);
    insert (root, 5);
}

```

```
inorder (root);  
return 0;
```

```
}
```

output:

37

43

51

98

3) ~~write a program for linear~~

3) write a c program depth first search (DFS) using a array.

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

```
void DFS (int);
```

```
int G[10][10], visited[10], n;
```

```
void main()
```

```
{
```

```
    int i, j;
```

```
    printf("Enter number of vertices:");
```

```
    scanf("%d", &n);
```

```
    printf("Enter adjacency matrix of the graph");
```

```
    for (i=0; i<n; i++)
```

```
        for (j=0; j<n; j++)
```

```
            scanf("%d", &G[i][j]);
```

```
        for (i=0; i<n; i++)
```

```
            visited[i]=0;
```

```
            DFS (0);
```

```
}
```

```
void DFS (int i)
```

```
{
```



```

    int j;
    printf ("%i\n", i);
    visited[i] = 1;
    for (j = 0; j < n; j++)
        if (!visited[j] && G[i][j] == 1)
            DFS(j);
}

```

Output:

Enter number of vertices: 6

Enter adjacency matrix of the graph: 1 0 1 0 0 1

1	0	1	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
0	1	0	1	0	1
1	0	1	0	1	0

0

2

1

4

3

5

4) write a C program breath first search (BFS) using array.

```
#include <stdio.h>
#include <stdlib.h>
#define Max 100
#define initial 1
#define waiting 2
#define visited 3
```

```
{
    int n;
    int adj[Max][Max];
    int state[Max];
    void create_graph();
    void BF_traversal();
    void BFS(int v);

    int queue[Max], front=-1, rear=-1;
    void insert_queue(int vertex);
    int delete_queue();
    int is_empty_queue();
    int main()
    {
        create_graph();
        BF_traversal();
        return 0;
    }

    void BF_traversal()
    {
        int v;
```

```

for (v=0; v<n; v++)
    state[v] = initial;
printf("Enter start vertex for BFS : ");
scanf("%d", &v);
BFS(v);
}

void BFS(int v)
{
    int i;
    insert_queue(v);
    state[v] = waiting;

    while (!is_empty_queue())
    {
        v = delete_queue();
        printf("%d", v);
        state[v] = visited;
        for (i=0; i<n; i++)
        {
            if (adj[v][i] == 1 && state[i] == initial)
            {
                insert_queue(i);
                state[i] = waiting;
            }
        }
    }
    printf("\n");
}

void insert_queue(int vertex)
{
    if (rear == MAX-1)

```



```
printf("Queue overflow\n");
```

```
else
```

```
{
```

```
if (front == -1)
```

```
front = 0;
```

```
rear = rear + 1;
```

```
queue[rear] = vertex;
```

```
}
```

```
}
```

```
int is_empty_queue()
```

```
{
```

```
if (front == -1 || front > rear)
```

```
return 1;
```

```
else
```

```
return 0;
```

```
}
```

```
int delete_item;
```

```
if (front == -1 || front > rear)
```

```
{
```

```
printf("Queue underflow\n");
```

```
exit
```

```
exit(1);
```

```
}
```

```
delete_item = queue[front];
```

```
front = front + 1;
```

```
return delete_item;
```

```
}
```

```
void create_graph()
```

```
{
```

```

int count, max-edge, origin, destin;
printf("Enter number of vertices");
scanf("%d", &n);
max-edge = n*(n-1);
for (count = 1; count <= max-edge; count++)
{
    printf("Enter edge %d (-1 -1 to quit):", count);
    scanf("%d %d", &origin, &destin);
    if ((origin == -1) && (destin == -1))
        break;
    if (origin >= n || destin >= n || origin < 0 || destin < 0)
    {
        printf("invalid edge !\n");
        count--;
    }
    else
    {
        adj[origin][destin] = 1;
    }
}
}
}

```

Output:

enter number of vertices: 9

enter edge 1 (-1 -1 to quit): 0

1

enter edge 2 (-1 -1 to quit): 0

3

enter edge 3 (-1 -1 to quit): 0

4

Enter edge 4 (-1 -1 to quit): 1
2

Enter edge 5 (-1 -1 to quit): 3
6

Enter edge 6 (-1 -1 to quit): 4
7

Enter edge 7 (-1 -1 to quit): 6
4

Enter edge 8 (-1 -1 to quit): 6
7

Enter edge 9 (-1 -1 to quit): 2
5

Enter edge 10 (-1 -1 to quit): 4
5

Enter edge 11 (-1 -1 to quit): 7
5

Enter edge 12 (-1 -1 to quit): 7
0

Enter edge 13 (-1 -1 to quit): 7
-1

Enter start vertex for BFS:
0

0 1 3 4 2 6 5 7

5.) write a program for linear search algorithm

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

```
int main()
```

```
{
```

```
    int array[100], search, c, n;
```

```
    printf("Enter number of elements in array\n");
```

```
    scanf("%d", &n);
```

```
    printf("Enter %d integers\n", n);
```

```
    for (c=0; c<n; c++)
```

```
        scanf("%d", &array[c]);
```

```
    printf("Enter a number to search\n");
```

```
    scanf("%d", &search);
```

```
    for (c=0; c<n; c++)
```

```
    { if (array[c] == search)
```

```
        {
```

```
            printf("%d present at location %d\n", search, c+1);
```

```
            break;
```

```
        }
```

```
    }
```

```
    if (c == n)
```

```
        printf("%d isn't present in the array\n", search);
```

```
    return 0;
```

```
}
```

output

Enter number of elements in array:

5

Enter 5 integers)

25

14

36

95

38

Enter a number to search

95

95 present at location 4.

6.) write a c program for binary search algorithm.

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

```
int main()
```

```
{ int c, first, last, middle, n, search, array[100];
```

```
printf("Enter number of elements\n");
```

```
scanf("%d", &n);
```

```
printf("Enter %d integers\n", n);
```

```
for (c=0; c<n; c++)
```

```
scanf("%d", &array[c]);
```

```
printf("Enter value to find\n");
```

```
scanf("%d", &search);
```

```
first = 0;
```

```
last = n-1;
```

```
middle = (first+last)/2;
```

```
while (first <= last)
```

```
{ if (array[middle] < search)
```

```
first = middle + 1;
```

```
else if (array[middle] == search)
```

```
{  
    printf("%d found at location %d\n", search,  
           middle + 1);
```

```
    break;
```

```
}
```

```
else
```

```
    last = middle - 1;
```

```
    middle = (first + last) / 2;
```

```
}
```

```
}
```

```
if (first > last)
```

```
    printf("not found! %d isn't present in  
           the list\n", search);
```

```
return 0;
```

```
}
```

Output:-

enter number of elements

5

enter 5 integers

25

36

41

51

95

enter value to find (n 51)

51 found at location 4.