Lab - 10 :- Hashing :-

```
#include <stdio.h>
#define TABLE_SIZE 10.

int hashFunction (int key) {
        return key % TABLE_SIZE;
    }

void insertValue (int hashTable[], int key)
 {
    int i=0;
    int hkey = hashFunction (key);
    int index;

 do {
    index = (hkey + i) % TABLE_SIZE;
     if (hashTable[index] == -1) {
        hashTable[index] = key
        printf ("Inserted key %d at index %d \n", key, index)
        return;
    }
    i++
 } while (i < TABLE_SIZE);

 printf ("Unable to insert key %d. Hash Table is full. \n", key);
}

int searchValue (int hashTable[], int key)
 {
    int i=0;
    int hkey = hashFunction (key);
    int index;
```

```
do {
    index = (hKey + i) % TABLE_SIZE;
    if (hashTable[index] == key) {
        printf("key %d found at index %d\n", key, index);
        return index;
    }
    i++;
} while (i < TABLE_SIZE);

printf("key %d not found in hash table\n", key);
    return -1;
}

int main() {
    int hashTable[TABLE_SIZE];
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i] = -1;
    }

    insertValue(hashTable, 18);
    insertValue(hashTable, 46);
    insertValue(hashTable, 7);
    insertValue(hashTable, 17);
    insertValue(hashTable, 1);

    return 0;
}
```

Output :-

```
Inserted key 18 at index 8
Inserted key 46 at index 6
Inserted key 7 at index 7
Inserted key 17 at index 9
Inserted key 1 at index 1
```

29/12/24

HackerRank :-

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

struct node
{
    int id;
    int depth;
    struct node *left, *
};

void
inorder(struct node *tre
{
    if (tree == NULL)
        return;

    inorder(tree->left);
    printf("%d ", tree->
    inorder((tree->ri
}

int main(void)
{
    int no_of_n
    int l, r, max_d
    struct node *t
    scanf("%d", &n
    struct node *tree
```