Harsh_Kumar_MAN_106_Assignment_6

- **Answer 1:**
  - Program :

```cpp
#include<bits/stdc++.h>
using namespace std;

#define MAX 100

class BST{
    int tree[MAX];
public :
    BST();
    void insert(int element);
    void traversal();
    void create(int *A, int n);
    int parent(int x);
    int left_child(int x);
    int right_child(int x);
    int search(int target);
    void tree_representation();
};

BST::BST(){
    for(int i=1; i<MAX; i++) tree[i] = -1;
    tree[0] = 0;
}

void BST::insert(int element){
    int root = 1;
    bool flag = false;
    for (int i=0; i<10; i++){
        if (tree[root] == -1) {
            tree[root] = element, flag = true;
            break;
        }
        if (tree[root] == element) break;
        else if (element < tree[root]){
            int left = 2*root;
            if (tree[left] != -1) root = left;
            else {
                tree[left] = element, flag = 1;
                break;
            }
        }
        else if (element > tree[root]){
            int right = 2*root + 1;
            if (tree[right] != -1) root = right;
            else {
                tree[right] = element, flag = 1;
                break;
            }
        }
    }

    if (flag) tree[0]++;

}

void BST::create(int *A, int n){
    for (int i=0; i<n; i++) insert(A[i]);
}
```

```cpp
void BST::traversal(){
    int cnt=0;
    cout << "Size = " << tree[0] << "\n";
    for(int i=1; i<MAX; i++) {
        cout << tree[i] << " ";
        if (tree[i] != -1) cnt++;
        if (cnt == tree[0]) break;
    }
    cout << '\n';
}

void BST::tree_representation(){
    int cnt = 0, j=1;
    cout << setw(40);
    for (int i=1; i<MAX; i++){
        cout << tree[i] << " ";
        if (tree[i] != -1) cnt++;
        if (i == pow(2, j) - 1) cout << "\n" << setw(40 - pow(2, j)), j++;
        if (cnt == tree[0]) break;
    }
    cout << "\n";
}

int BST::parent(int x) { return tree[x/2]; }
int BST::left_child(int x) { return tree[2*x]; }
int BST::right_child(int x) { return tree[2*x+1]; }

int BST::search(int target){
    int root = 1;
```

```cpp
    for (int i=0; i<10; i++){
        if (tree[root] == -1) {
            cout << "search unsuccesful ! key not present\n";
            return -1;
        }
        if (tree[root] == target) {
            cout << "search succesful\n";
            return root;
        }
        else if (target < tree[root]){
            int left = 2*root;
            if (tree[left] != -1) root = left;
            else {
                cout << "search unsuccesful ! key not present\n";
                return -1;
            }
        }
        else if (target > tree[root]){
            int right = 2*root + 1;
            if (tree[right] != -1) root = right;
            else {
                cout << "search unsuccesful ! key not present\n";
                return -1;
            }
        }
    }

    cout << "search unsuccesful ! key not present\n";
    return -1;
}

```

```cpp
int main(){

    BST binary_tree;
    int A[] = {40, 30, 50, 20, 10, 60, 90, 80, 100, 70};
    binary_tree.create(A, 10);

    binary_tree.traversal();
    binary_tree.tree_representation();

    binary_tree.insert(15);
    binary_tree.tree_representation();

    int found = binary_tree.search(15);
    cout << "15 found at " << found << "\n";

    cout << binary_tree.parent(found) << " " << binary_tree.left_child(found) << " " << binary_tree.right_child(found) << "\n";

    return 0;
}
```

- Output :

```
Size = 10
40 30 50 20 -1 -1 60 10 -1 -1 -1 -1 -1 -1 90 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 80 100 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
    -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 70
                                      40
                                    30 50
                                  20 -1 -1 60
                              10 -1 -1 -1 -1 -1 -1 90
                      -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 80 100
      -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 70
                                      40
                                    30 50
                                  20 -1 -1 60
                              10 -1 -1 -1 -1 -1 -1 90
                      -1 15 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 80 100
      -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 70
search succesful
15 found at 17
10 -1 -1
```

- **Answer 2:**
  - Program :

```cpp
#include<bits/stdc++.h>
using namespace std;

struct node{
    int data;
    node* left_child;
    node* right_child;

    node(int val, node* left_child = NULL, node* right_child = NULL): data(val), left_child(NULL), right_child(NULL){}
};

class BinartTree{
    node* ROOT;
public:
    void makeTree(int *A, int n);
    void print_tree();
    int depth(node *A);
    int find_depth();
};

void BinartTree::makeTree(int *A, int n){
    queue<node *> q;
    ROOT = new node(A[0]);
    q.push(ROOT);
    for( int i=0; i<n; ){
        auto prev = q.front();
        q.pop();
        prev->left_child = new node(A[++i]);
        prev->right_child = new node(A[++i]);
        q.push(prev->left_child);
        q.push(prev->right_child);
    }
```

```
void BinartTree::print_tree(){

    if (ROOT == NULL) return;

    stack<node *> s;
    s.push(ROOT);

    while (s.empty() == false){
        auto temp = s.top();
        s.pop();
        cout << temp->data << " ";
        if (temp->right_child) s.push(temp->right_child);
        if (temp->left_child) s.push(temp->left_child);
    }
    cout << "\n";
}

int BinartTree::depth(node *A){
    if (A == NULL) return 0;
    return max(depth(A->left_child)+1, depth(A->right_child)+1);
}

int BinartTree::find_depth(){
    return depth(ROOT);
}

int main(){
    BinartTree binary_tree;
    int A[] = {10, 20, 50, 30, 60, 80, 70, 90, 100, 40, 110};

    binary_tree.makeTree(A, 10);
```

```

/*
    Tree made :
                    10
                20          50
            30      60      80   70
        90   100  40  110

*/
    cout << "Pre-order traversal:\n";
    binary_tree.print_tree();

    cout << "\nDepth = " << binary_tree.find_depth();
}
```

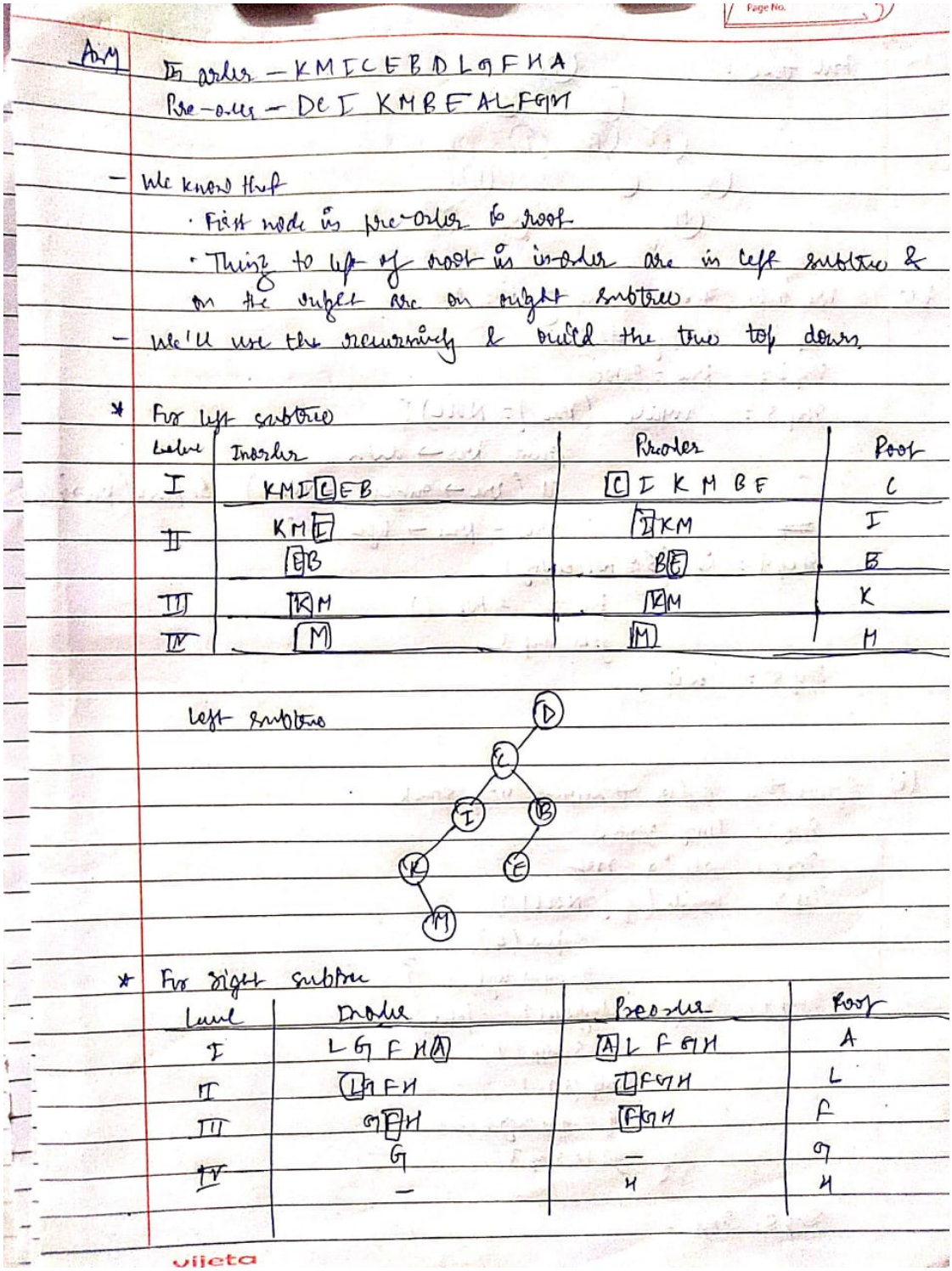
- Output :

```
Pre-order traversal:
10 20 30 90 100 60 40 110 50 80 70

Depth = 4
```

- **Answer 3:**
  - Program:

```
int BinartTree::depth(node *A){
    if (A == NULL) return 0;
    return max(depth(A->left_child)+1, depth(A->right_child)+1);
}

int BinartTree::find_depth(){
    return depth(ROOT);
}
```

- **Answer 4 and 5 :**



Ans

In order – K M I C E B D L G F H A

Pre-order – D C I K M B E A L F G H

– We know that

· First node in pre-order is root.

· Things to left of root in in-order are in left subtree & on the right are on right subtree.

– We'll use the recursively & build the tree top down.

* For left subtree

Level	Inorder	Preorder	Root
I	K M I [C] E B	[C] I K M B E	C
II	K M [I]	[I] K M	I
	[E] B	B [E]	E
III	[K] M	[K] M	K
IV	[M]	[M]	M

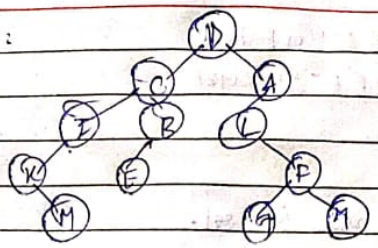
Left subtree

* For right subtree

Level	Inorder	Preorder	Root
I	L G F H [A]	[A] L F G H	A
II	[L] G F H	[L] F G H	L
III	G [F] H	[F] G H	F
IV	G	–	G
	–	H	H

vijeta

Full tree :



Ans (a) Pre-order Traversal using Stack

Step 1: Empty stack S.

Step 2: pres = root

Step 3: while (pres != NULL) {
print pres → data;
if (pres → right != NULL) S.push(pres→right)
pres = pres → left; }

Step 4: if (S is not empty)
pres = S.top()
goto step 3

Step 5: exit.

(b) Symmetric order traversal using stack

Step 1: Empty stack S

Step 2: node *q = root

Step 3: while (q != NULL) {
S.push(q)
q = q → left; }

Step 4: if (S.underflow() = false)
q = S.pop();
print(q);
q = q → right;
go to step 3
}

Step 5: exit.

vijeta

(1) Level order traversal

Step 1: Init q queue q.

Step 2: q.push(root)

Step 3: while (q.underflow() == false)

node * x = q.pop()

print(x.info);

if (x->left) q.push(x->left)

if (x->right) q.push(x->right)

}

Step 4: exit.