Lab 6

Due No Due Date **Points** 1

You are suggested to use the teaching servers burrow.soic.indiana.edu or hulk.soic.indiana.edu or tank.soic.indiana.edu for practicing C programs.

Lab 6: Binary Tree

Take an array of size 'n' and insert in Binary Search Tree(BST) one by one. The elements of array should be randomly generated. Do an in-order traversal of BST to print the array in ascending order. For this lab, all students must have a good grasp on pointer and linked list traversing.

Binary Tree

A binary tree is a tree data structure in which each node has **at most** two children, which are referred to as the left child and the right child. The **left** child value must be **less than** the parent node, and **right** child must be **equal or greater to** the parent node. Nodes without any child node is called a leaf node. There are no parent of the root node. The first inserted node is considered as the root node.

Pseudo Code:

We can define a node with an integer value and two left and right child pointer as below:

```
typedef struct node_struct
{
        int value;
        struct node_struct *left;
        struct node_struct *right;
} node;
```

The tree can be initiated like this

```
node *root;
root = NULL;
```

We can search for a value in a tree in O(h) time, where h is the height of the time. The runtime will be significantly lower for an fairly **balanced** tree. We start the search in the root node and traverse down the tree comparing the tree node value with our desired value.

```
TREE-SEARCH(node *p, value)  //return the node found or NULL if not found
  if p == NULL
        return NULL
  if value == p->value
        return p
  if value < p->value
        return TREE-SEARCH(p->left, value)
  else
        return TREE-SEARCH(p->right, value)
```

There is also an iterative version.

```
TREE-SEARCH-ITERATIVE(node *p, value)
     while(p != NULL && p->value != value){
          if (value < p->value)
              p = p->left
          else
              p = p->right
      return p
```

Pseudo code for insertion:

```
 void insert(node **p, int value)
{
        node *temp = NULL;
        if(*p == NULL)
        {
                temp = (node *)malloc(sizeof(node));
                temp->left = temp->right = NULL;
                temp->value = value;
                *p = temp;
                return;
        }

        if(value < (*p)->value)
                insert(&((*p)->left), value);
        else
                insert(&((*p)->right), value);

}
```