Assignment 6: BSTADT and its application

Create an ADT for the binary search tree data structure with the following functions. Each node which consists of integer data, address of left and right children.

a. insertBST(t,data) – insert data into BST
b. inorder(t) – display the tree using inorder traversal
c. preorder(t) – display the tree using preorder traversal
d. postorder(t) – display the tree using postorder traversal
e. levelorder(t) – display the tree hierarchically
f. findmin(t)– returns the minimum element in the tree
g. search(t,key) – returns the element found, otherwise returns NULL
h. delete(t,elt) – delete the given elt from tree

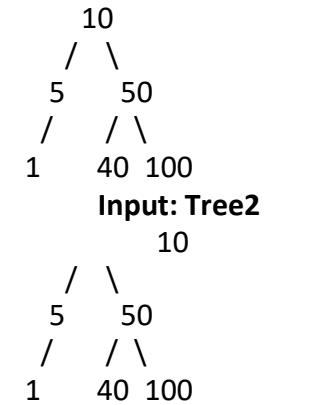
1. Demonstrate the BSTADT with the following test case

   Insert(t,29)
   Insert(t,23)
   Insert(t,4)
   Insert(t,13)
   Insert(t,39)
   Insert(t,31)
   Insert(t,45)
   Insert(t,56)
   Insert(t,49)
   Inorder(t) → 4,13,23,29,31,39,45,49,56
   Levelorder(t)→ 1st Level → 29
   2nd level → 23, 39
   3rd Level →4, 31, 45
   4th Level →13, 56
   5th Level → 49
   Findmin(t) → 4
   Find(t, 13) → Found, value is 3
   Find(t,3) → Not found

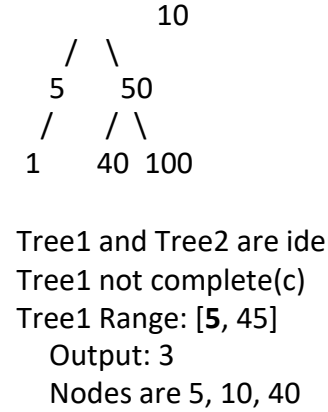
2. Write an application to do the following
   a. Check whether the two BST contains the same set of elements
   b. Count the number of nodes in tree within the given range
   c. Find sum of k smallest elements in the given BST

Test case for the Application(a)

Input: Tree 1



Input: Tree2



Tree1 and Tree2 are identical with a set of elements(b)

Tree1 not complete(c)

Tree1 Range: [**5**, 45]

Output: 3

Nodes are 5, 10, 40

Tree2 Range: [**1**, 45]

Output: 4

Nodes are 1,5, 10, 40

Given two binary search trees `root1` and `root2`, return *a list containing all the integers from both trees sorted in* ***ascending*** *order*.

CODE:

BSTADT.h

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

struct tree {
   int data;
   struct tree* left;
   struct tree* right;
};

struct queue {
   struct tree* data;
   struct queue* next;
};
```

```
struct tree* insert(struct tree* t, int x) {
   if (t == NULL) {
      t = (struct tree*)malloc(sizeof(struct tree));
      t->data = x;
      t->left = NULL;
      t->right = NULL;
   } else if (x < t->data) {
      t->left = insert(t->left, x);
   } else if (x > t->data) {
      t->right = insert(t->right, x);
   }
   return t;
}

void inorder(struct tree* t) {
   if (t != NULL) {
      inorder(t->left);
      printf("%d ", t->data);
      inorder(t->right);
   }
}

void preorder(struct tree* t) {
   if (t != NULL) {
      printf("%d ", t->data);
      preorder(t->left);
      preorder(t->right);
   }
}

void postorder(struct tree* t) {
   if (t != NULL) {
      postorder(t->left);
      postorder(t->right);
      printf("%d ", t->data);
   }
}

void enq(struct tree* data, struct queue** q) {
```

```
    struct queue* newNode = (struct queue*)malloc(sizeof(struct queue));
    newNode->data = data;
    newNode->next = NULL;

    if (*q == NULL) {
        *q = newNode;
    } else {
        struct queue* temp = *q;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

void deq(struct queue** q) {
    if (*q != NULL) {
        struct queue* temp = *q;
        *q = (*q)->next;
        free(temp);
    }
}

void levelorder(struct tree* t) {
    struct queue* q = NULL;

    if (t == NULL) {
        return;
    }

    enq(t, &q);

    while (q != NULL) {
        struct tree* current = q->data;
        printf("%d ", current->data);

        if (current->left != NULL) {
            enq(current->left, &q);
        }
```

```
      if (current->right != NULL) {
         enq(current->right, &q);
      }

      deq(&q);
   }
}


struct tree* search(struct tree* t, int x) {
   if (t == NULL) {
      return NULL;
   } else if (x == t->data) {
      return t;
   } else if (x < t->data) {
      return search(t->left, x);
   } else {
      return search(t->right, x);
   }
}

int findmin(struct tree* t) {
   if (t->left != NULL) {
      return findmin(t->left);
   }
   return t->data;
}

struct tree* deletenode(struct tree* t, int x) {
   if (t == NULL) {
      return t;
   }

   if (x < t->data) {
      t->left = deletenode(t->left, x);
   } else if (x > t->data) {
      t->right = deletenode(t->right, x);
   } else if (t->left && t->right) {
      int temp = findmin(t->right);
      t->data = temp;
```

```
        t->right = deletenode(t->right, temp);
    } else {
        struct tree* temp = t;
        if (t->right == NULL) {
            t = t->left;
        } else if (t->left == NULL) {
            t = t->right;
        }
        free(temp);
    }
    return t;
}
```

Main.h

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

int main() {
    struct tree* t;
    t = NULL;
    t = insert(t, 2);
    insert(t, 3);
    insert(t, 4);
    insert(t, 1);
    insert(t, 36);
    insert(t, 49);

    int Search = 4;
    struct tree* temp = search(t, Search);

    if (temp == NULL) {
        printf("Element %d is not found\n", Search);
    } else {
        printf("Element %d is found\n", Search);
    }

    printf("Inorder: ");
    inorder(t);
    printf("\nPreorder: ");
```

```
    preorder(t);
    printf("\nPostorder: ");
    postorder(t);
    printf("\nLevelorder: ");
    levelorder(t);
    printf("\n");

    printf("\nFindMin: ");
    int a = findmin(t);
    printf("%d", a);

    printf("\nAfter deleting 13: ");
    deletenode(t, 13);

    printf("\nInorder of traversal: ");
    inorder(t);
    printf("\nPreorder of traversal: ");
    preorder(t);
    printf("\nPostorder of traversal: ");
    postorder(t);

    return 0;
}
```

Output:

```
Element 4 is found
Inorder: 1 2 3 4 36 49
Preorder: 2 1 3 4 36 49
Postorder: 1 49 36 4 3 2
Levelorder: 2 1 3 4 36 49

FindMin: 1
After deleting 13:
Inorder of traversal: 1 2 3 4 36 49
Preorder of traversal: 2 1 3 4 36 49
Postorder of traversal: 1 49 36 4 3 2
```

2.Write an application to do the following

a. Check whether the two BST contains the same set of elements
b. Count the number of nodes in tree within the given range
c. Find sum of k smallest elements in the given BST

Algorithm:

Code:

a. Check whether the two BST contains the same set of elements

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct tree {

int data;

struct tree *right; struct tree *left;

};


struct tree *insert(struct tree *t, int x) { if (t == NULL) {

t = (struct tree *)malloc(sizeof(struct tree)); t->data = x;

t->right = NULL; t->left = NULL;

} else if (x > t->data) {

t->right = insert(t->right, x);

} else if (x < t->data) {

t->left = insert(t->left, x);

}

return t;

void inorder(struct tree *t, int a1[], int *i) {

if (t==NULL) {

return;

}

inorder(t->left, a1, i); printf("%d\t", t->data);
```

```
a1[(*i)++] = t->data; inorder(t->right, a1, i);
}

        int findsome(int a1[], int a2[], int *i, int *j) {
 int flag = 0;
if (*i != *j) { return -1;
}
else {
for (int k = 0; k < *j; k++) {
 if (a1[k] != a2[k]) {
return -1;
}
}
}
return 1;
}
int main() {
struct tree *root1; root1 = NULL; struct tree *root2; root2 = NULL;
root1 = insert (root1, 29); insert(root1, 23);
insert(root1, 4);
insert(root1, 13);
insert(root1, 39);
insert(root1, 31);
insert(root1, 45);
insert(root1, 56);
insert(root1, 49);
root2 = insert(root2, 29); insert(root2, 23);
insert(root2, 4);
```

```
insert(root2, 13);
insert(root2, 39);
insert(root2, 31);
insert(root2, 45);
insert(root2, 56);
insert(root2, 49);
int a1[100];
int a2[100];
int i = 0;
int j=0;
printf("Inorder(t) -> ");
inorder(root1, a1, &i);
printf("\n");
printf("Inorder(t) -> ");
inorder(root2,a &j);
printf("\n");
int res = findsome(a1, a2, &i, &j);
 if (res == -1) {
printf("THE BOTH TREES CONTAINS 'DIFFERENT' ELEMENTS");
}
else {
printf("THE BOTH TREES CONTAINS 'SAME' ELEMENTS");
}
return 0;
}
```

Output:

```
Inorder(t) -> 4 13  23  29  31  39  45  49
Inorder(t) -> 4 13  23  29  31  39  45  49
THE BOTH TREES CONTAINS 'SAME' ELEMENTS~/tr
```

b. Count the number of nodes in tree within the given range

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
struct tree
{
 int data;
 struct tree *left;
 struct tree *right;
};
struct tree *insert(struct tree *t,int x)
{
 if(t==NULL)
 {
 t=(struct tree *)malloc(sizeof(struct tree));
 t->data=x;
 t->right=NULL;
 t->left=NULL;
 }
 else if(x<t->data)
 {
 t->left=insert(t->left,x);
 }
 else if(x>t->data)
 {
 t->right=insert(t->right,x);
 }
```

```
 return t;
}
int findcounts(struct tree *t,int l ,int h)
{
 if(t==NULL)
 {
 return 0;
 }
 if(t->data >=l && t->data<= h)
 {

 return 1 + findcounts(t->right,l,h)+findcounts(t->left,l, h);
 }
 else if(t->data < l)
 {
 return findcounts(t->right,l,h);
 }
 else if(t->data > h )
 {
 return findcounts(t->left,l,h);
 }
}
void inorder(struct tree *t)
{
 if (t == NULL) {
 return;
 }
 inorder(t->left);
```

```
 printf("%d\t", t->data);
 inorder(t->right);
}

int main()
{
 struct tree *root1;
 root1 = NULL;
 root1 = insert(root1, 29);
 insert(root1, 23);
 insert(root1, 4);
 insert(root1, 13);
 insert(root1, 39);
 insert(root1, 31);
 insert(root1, 45);
 insert(root1, 56);
 insert(root1, 49);
 inorder(root1);
 int res=findcounts(root1, 0, 40 );
 printf("\nThe number of nodes within the given range is %d:",res);
}
```

Output:

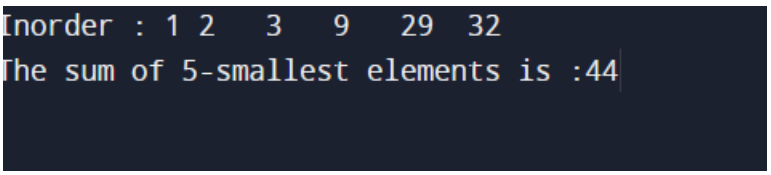
```
4   13  23  29  31  39  45  49  56
The number of nodes within the given range is 6:
```

c. #include<stdio.h>
#include<stdlib.h>
#include<math.h>
struct tree
{
 int data;
 struct tree *left;

```
 struct tree *right;
};
struct tree *insert(struct tree *t,int x)
{
 if(t==NULL)
 {
 t=(struct tree *)malloc(sizeof(struct tree));
 t->data=x;
 t->right=NULL;
 t->left=NULL;
 }
 else if(x<t->data)
 {
 t->left=insert(t->left,x);

 }
 else if(x>t->data)
 {
 t->right=insert(t->right,x);
 }
 return t;
}
void inorder(struct tree *t, int a1[], int *i) {
 if (t == NULL) {
 return;
 }
 inorder(t->left, a1, i);
 printf("%d\t", t->data);
 a1[(*i)++] = t->data;
 inorder(t->right, a1, i);
}
int findsum(struct tree *t,int a[],int k)
{
 int sum =0;
 for(int l=0;l<k;l++)
 {
 sum=sum+a[l];
 }
 return sum;
}
int main()
{
 struct tree *root1;
 root1 = NULL;
 root1 = insert(root1, 29);
 insert(root1, 2);
 insert(root1, 3);
 insert(root1, 1);
 insert(root1, 9);
 insert(root1, 32);

 int a1[100];
 int i = 0;
```

```
 printf("Inorder : ");
 inorder(root1, a1, &i);
 printf("\n");

 int res3= findsum(root1, a1, 5);
 printf("The sum of 5-smallest elements is :%d",res3);
}
```

Output:



Learning Outcome:

I learnt about binary search tree .performing insertion, deletion, find minimum, count, search of basic function of Binary search tree.