浙江大學

Fundamentals of Data Structures
Laboratory Project 2

BUILD A BINARY SEARCH TREE PROBLEM

COMPLETE DATE: 2018/11/02

CONTENTS

Chapter 1

Introduction

1.1 Background

In class, we have learned some knowledge about Binary Search Trees. And we know that a Binary Search Tree is either an empty tree or a binary tree with the following properties:

- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than or equal to the node's key.
- Both the left and right subtrees must also be binary search trees.

In this project, our team tried to build a binary search tree and tested it with a series of data.

1.2 Algorithm requirements

In this Algorithm, we need to give the structure of a binary tree and a sequence of distinct integer keys, there is only one way to fill these keys into the tree so that the resulting tree satisfies the definition of a BST. You are supposed to output the level order traversal sequence of that tree.

The Input Specification is that: Each input file contains one test case. For each case the first line gives a positive integer N (N≤ 100) which is the total number of nodes in the tree. The next N lines each contains the left and the right children of a node in the format left_index right_index provided that the nodes are numbered from 0 to N – 1 and 0 is always the root. If one child is missing then −1 will represent the NULL child pointer. Finally N distinct integer keys are given in the last line.

Output Specification is that: For each test case, print in one line the level order traversal sequence of that tree. All the numbers must be separated by a space with no extra space at the end of the line.

1.3 Achieve results

Within the effective time, our team successfully wrote a program to build a binary search tree, and completed the effective annotation of the program.

After the programmer completes all the program requirements, the tester tests the tree through a series of data. All the data tested conforms to the definition of binary search tree.

Chapter 2

Algorithm Specification

Our idea to solve this problem is that built a binary tree according to the input first, then assign the elements in the number to make it a binary search tree, and finally traverse the output sequence.

2.1 Algorithm for building a tree

In this function, we set up a tree in the form of an array; we define two structural variables, and define two arrays with these two nodes. Array T refers to the tree, and array info refers to the sub-node of the node corresponding to info[i]. If info[i], the value of a parameter is -1, then it denotes its sub-node. Points do not exist, if it is other integers, it indicates that a sub-node of the node represented by info[i] points to the node corresponding to the integer.

```
Tree BuildTree(int N, struct Information *info)
{
    Tree T[N];
    int i;
    for(i=0;i<N;i++)
        T[i] = (Tree)malloc(sizeof(struct Node));
    for(i=0;i<N;i++)
    {
        if(info[i].left==-1)
            T[i]->Left = NULL;
        else
            T[i]->Left = T[info[i].left];
        if(info[i].right==-1)
            T[i]->Right = NULL;
        else
            T[i]->Right = T[info[i].right];
    }
    return T[0];
}
```

2.2 Algorithm for inorder traversal

The middle order traversal of a binary search tree must be output in a certain order. Here, the output is from small to large. Considering the characteristics of the binary search tree, we can use the idea of middle order traversal to input data into the binary search tree in this order.

We store the number we want to input in an array, and then enter the number in the array into the binary search tree in the order of intermediate traversal.

```
void InOrder (Tree T, int *num)
{
    if(T)
    {
        InOrder(T->Left, num);
        T->key = num[count++];
        InOrder(T->Right, num);
    }
}
```

2.3 Algorithm for levelorder traversal

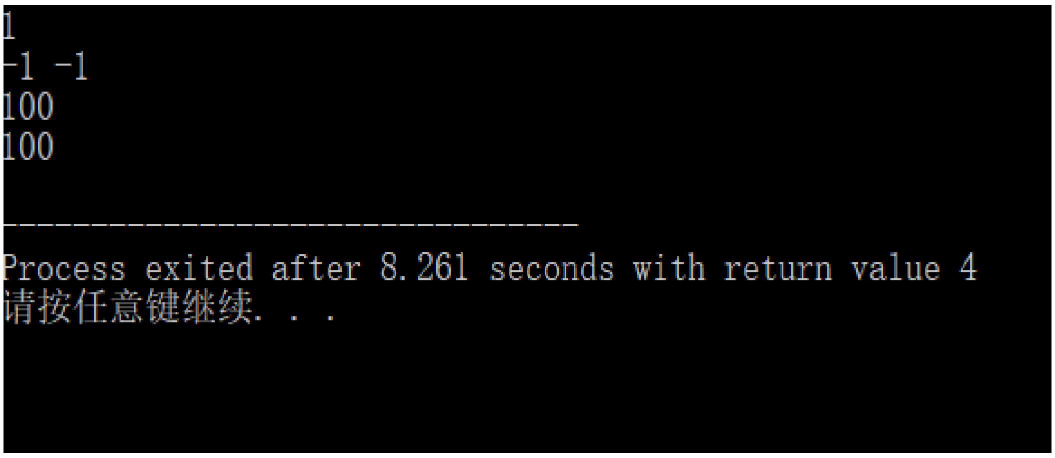
The last function is levelorder traversal. According to the output specification, we use the method of levelorder traversal to output the elements in the tree layer by layer.

```
void LevelOrderPrint(Tree T)
{
    Tree queue[100];
    int rear=0, i=0, temp=0;
    if(!T)
        return;
    else
    {
        queue[rear] = T;
        rear++;
        while(temp!=rear)
        {
            if(queue[temp]->Left)
            {
                queue[rear] = queue[temp]->Left;
                rear++;
            }
            if(queue[temp]->Right)
            {
                queue[rear] = queue[temp]->Right;
                rear++;
            }
            temp++;
        }
        for(i=0;i<rear-1;i++)
            printf("%d ",queue[i]->key);
        printf("%d\n",queue[i]->key);
        return;
    }
}
```

Chapter 3

Testing Results

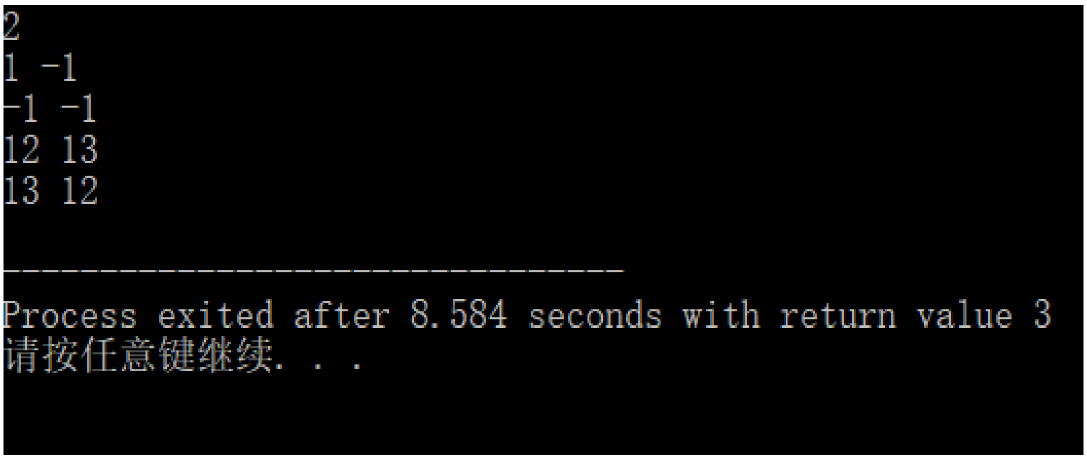
1. When there is only one root node, the test result is correct.



```
1
-1 -1
100
100

--------------------------------
Process exited after 8.261 seconds with return value 4
请按任意键继续. . .
```

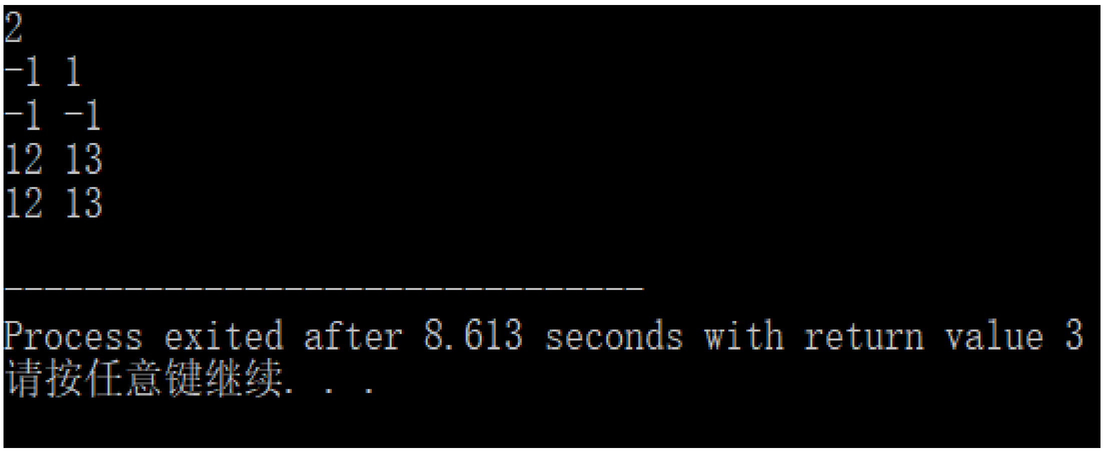
2. there are only two nodes, and the second nodes are left nodes. The test results are correct.



```
2
1 -1
-1 -1
12 13
13 12

--------------------------------
Process exited after 8.584 seconds with return value 3
请按任意键继续. . .
```

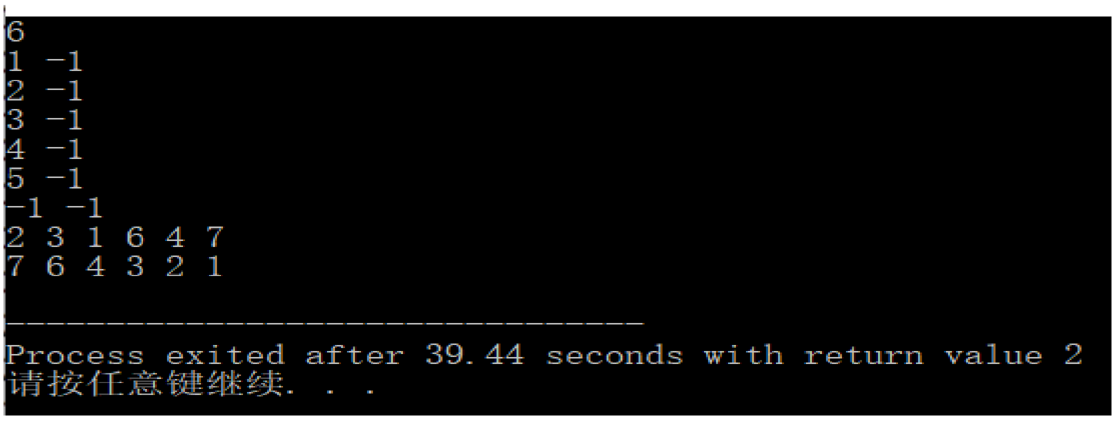
3. there are only two nodes, and the second nodes are the right nodes. The test result is correct.



```
2
-1 1
-1 -1
12 13
12 13

--------------------------------
Process exited after 8.613 seconds with return value 3
请按任意键继续. . .
```

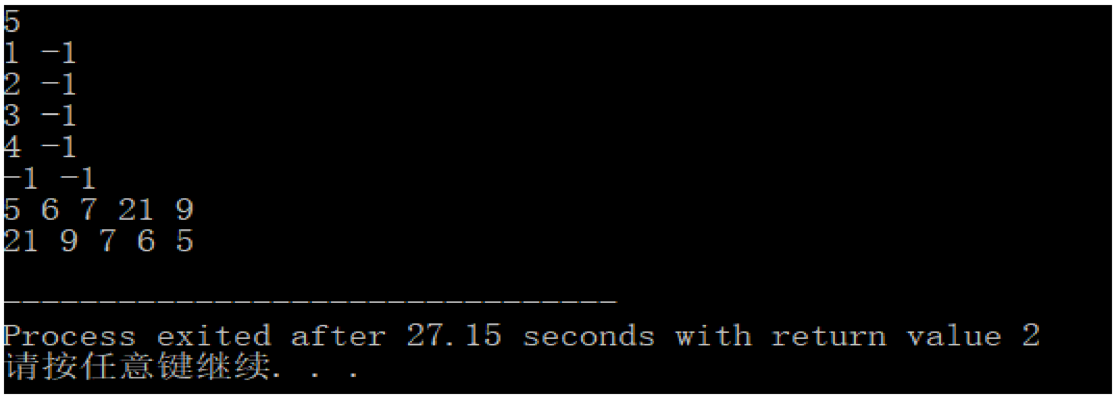
4. Trees containing only left nodes are correct, and the test result is correct.



```
6
1 -1
2 -1
3 -1
4 -1
5 -1
-1 -1
2 3 1 6 4 7
7 6 4 3 2 1

--------------------------------
Process exited after 39.44 seconds with return value 2
请按任意键继续. . .
```

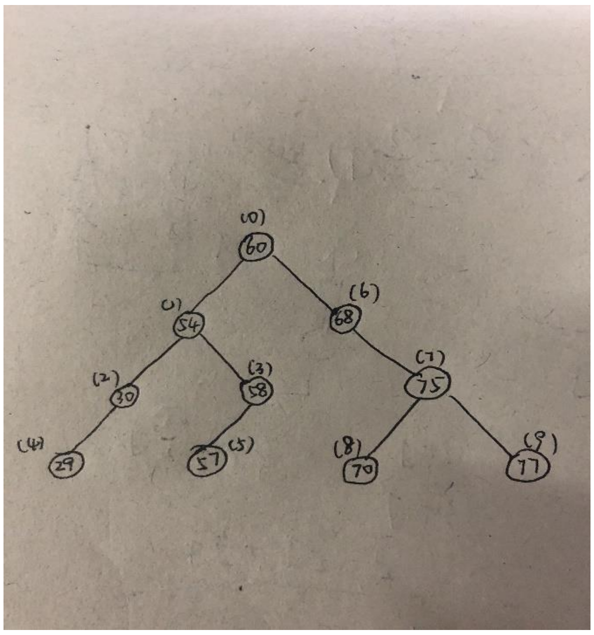
5. Trees containing only the right nodes are correct, and the test results are correct.

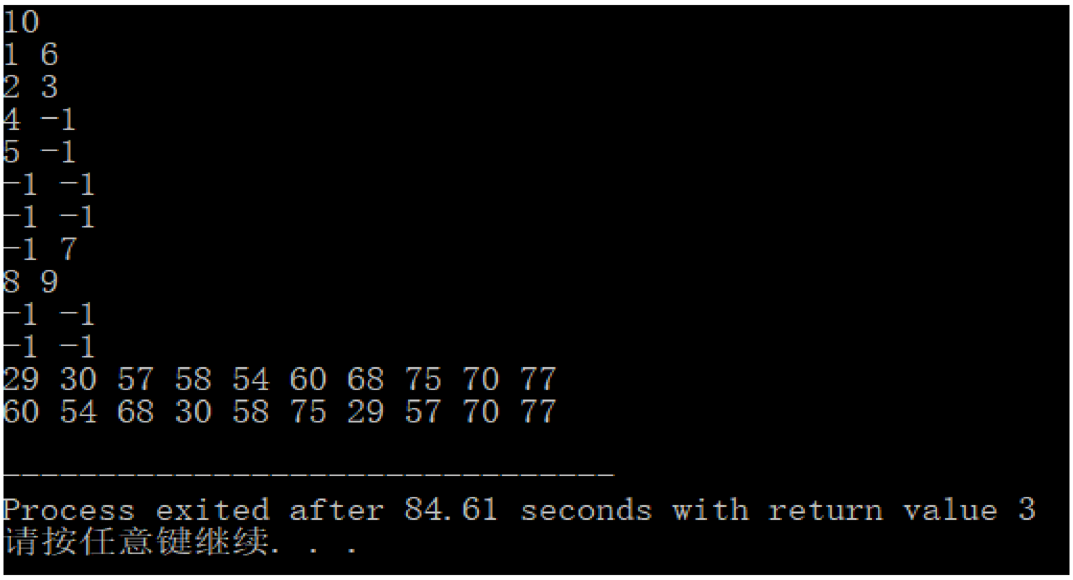


```
5
1 -1
2 -1
3 -1
4 -1
-1 -1
5 6 7 21 9
21 9 7 6 5

--------------------------------
Process exited after 27.15 seconds with return value 2
请按任意键继续. . .
```

6.equivalent to sample, correct test result.

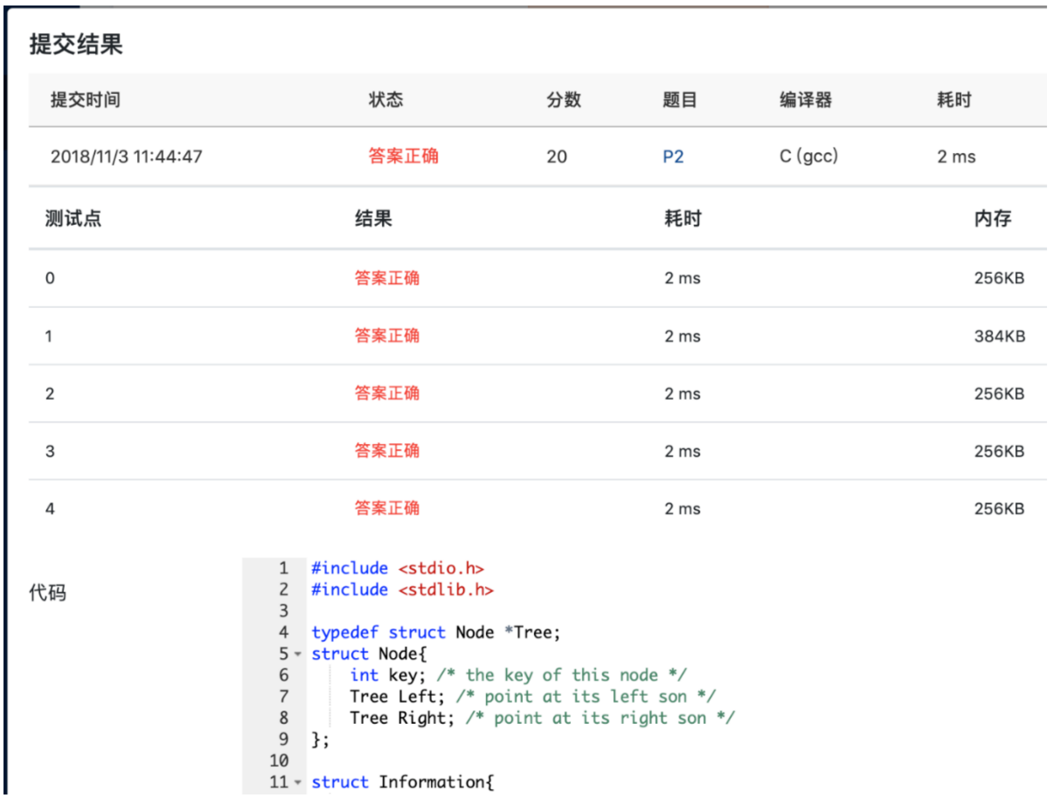




```
10
1 6
2 3
4 -1
5 -1
-1 -1
-1 -1
-1 7
8 9
-1 -1
-1 -1
29 30 57 58 54 60 68 75 70 77
60 54 68 30 58 75 29 57 70 77

--------------------------------
Process exited after 84.61 seconds with return value 3
请按任意键继续. . .
```

7.On PTA, the test results are all correct.



提交结果

提交时间	状态	分数	题目	编译器	耗时
2018/11/3 11:44:47	答案正确	20	P2	C (gcc)	2 ms

测试点	结果	耗时	内存
0	答案正确	2 ms	256KB
1	答案正确	2 ms	384KB
2	答案正确	2 ms	256KB
3	答案正确	2 ms	256KB
4	答案正确	2 ms	256KB

代码

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

typedef struct Node *Tree;
struct Node{
    int key; /* the key of this node */
    Tree Left; /* point at its left son */
    Tree Right; /* point at its right son */
};

struct Information{
```

Chapter 4

Analysis and Comments

Analysis

In this project, the code the programmer writes has the beginning structure definition section, a main function, and three functions

Definition of structure and structure pointer: At the beginning of the program, tree node pointer Tree is defined, which is mainly used to point to the node of the tree, so that a tree can be constructed. Then, a structure information is defined to store the information of the left and right sons of each node. The order in which an element traverses first.

Function part: Define a BuildTree function prototype Tree BuildTree (int N, struct Information * info), this function establishes an array of pointers, each element type is Tree, and through the info array of information to build a tree, and each element in the tree for the node first The order of ordered traversal is followed by the definition of a function prototyped void InOrder (Tree T, int * num), which assigns elements in the array num that stores node elements to each node of the established tree. Since num is already ordered, it is only necessary to assign the whole tree in the order of ordered traversal to achieve its goal. The last function is void LevelOrderPrint (Tree T), which is used to output the result, using a queue to sequence the key tree and output the result of the traversal.

The main function part: The main operation of the main function is to read in the data, including the number of nodes, the sequence of nodes, the value of node elements, and the array of storage node elements from small to large sort, convenient for later operation, the main function finally called three functions, thus achieving The whole procedure.

Next, the algorithm complexity of the whole program is analyzed. The highest level of the whole cycle is to sort all the elements in the tree. Bubble sorting is used. There are two levels of loops. Therefore, the time complexity of the whole program is O(N2). And the whole program only uses a recursive algorithm in one place, that is, to use the tree. During the sequential traversal, the space complexity of the whole algorithm is O (N).

Comment

The whole program is very clear and organized, contains the idea of modular problem solving, is also conducive to other people's understanding, at the same time, the code has enough and detailed comments, reflects the programmer's thinking process and ideas, and makes it easier for readers to understand the whole program thoroughly.

On the disadvantage, I think sorting algorithm can use less time complexity algorithm, and each time a new variable is defined, it is better to explain the role of this variable.

Appendix

Source Code (in C)

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

typedef struct Node *Tree;
struct Node{
    int key; /* the key of this node */
    Tree Left; /* point at its left son */
    Tree Right; /* point at its right son */
};

struct Information{
    int left;   /* index of the number of left son */
    int right;   /* index of the number of right son */
};

Tree BuildTree(int N, struct Information *info) /* build a new tree */
{
    Tree T[N]; /* define a tree with N nodes */
    int i;
    for(i=0;i<N;i++)   /* allocate space for the tree */
        T[i] = (Tree)malloc(sizeof(struct Node));
    for(i=0;i<N;i++)
    {
        if(info[i].left==-1)   /* has no left son */
```

```
                T[i]->Left = NULL;
            else
                T[i]->Left = T[info[i].left];    /* point at info[i].left */
            if(info[i].right==-1)    /* has no right son */
                T[i]->Right = NULL;
            else
                T[i]->Right = T[info[i].right];    /* point at info[i].right */
        }
        return T[0];    /* return the root */
}

int count = 0;
void InOrder (Tree T, int *num)    /* inorder traversal as well as input keys */
{
        if(T) /* not empty */
        {
            InOrder(T->Left, num);    /* inorder traversal the left subtree */
            T->key = num[count++];    /* input the sorted key into the node */
            InOrder(T->Right, num);    /* inorder traversal the right subtree */
        }
}

void LevelOrderPrint(Tree T)    /* level-order traversal as well as output */
{
        Tree queue[100];    /* create a queue to save all the nodes */
        int rear=0, i=0, temp=0;
        if(!T)    /* empty tree */
            return;
        else
        {
            queue[rear] = T;
            rear++;
            while(temp!=rear)
            {
                if(queue[temp]->Left)
```

```
				{
					queue[rear] = queue[temp]->Left;
					rear++;
				}
				if(queue[temp]->Right)
				{
					queue[rear] = queue[temp]->Right;
					rear++;
				}
				temp++;
			}
			for(i=0;i<rear-1;i++)    /* output the level order traversal sequence of
the tree */
				printf("%d ",queue[i]->key);
			printf("%d\n",queue[i]->key);
			return;
		}
}

int main()
{
	int N;    /* the total number of nodes in the tree */
	int i,j,temp;
	scanf("%d\n",&N);
	int num[N];    /* save all the keys */
	struct Information info[N]; /* save all the relation information of each node
*/

	for(i=0;i<N;i++) /* input all the information of each node */
	{
		scanf("%d%d\n",&info[i].left,&info[i].right);
	}
	for(i=0;i<N;i++) /* input all the keys into an array */
	{
		scanf("%d",&num[i]);
```

```
    }
    for(i=0;i<N;i++)    /* bubble sort the array */
    {
        for(j=0;j<N-1;j++)
        {
            if(num[j]>num[j+1])
            {
                temp = num[j];
                num[j] = num[j+1];
                num[j+1] = temp;
            }
        }
    }
    /* the sequence of sorted array is the same as inorder traversal of BST */
    Tree BST = BuildTree(N, info); /* build a tree according to the information
*/
    InOrder(BST, num);    /* inorder traversal and input the sorted keys into
the tree */
    LevelOrderPrint(BST);    /* print the keys in level order */
}
```

Declaration

We hereby declare that all the work done in this project titled " BUILD A BINARY SEARCH TREE " is of our independent effort as a group.

Duty Assignments:

Programmer: XXX
Tester: XXX
Report Writer: XXX