**BST Sort**

**Done By:** Rohit Karunakaran **Roll No:** 58

**Aim:** Sort a array of numbers using binary search tree

**Data Structures used:** Linked List, Binary Tree, Array

**Algorithm for Insertion**

**Input:** The root node (root) and the key, element to be inserted

**Output :** The binary search tree with the node inserted

**Data Structure :** Binary Search Tree

**Steps**

* 1. Step 1: Start
  2. Step 2: ptr = root
  3. Step 3: while(ptr!=NULL and flag==true) do
  4. Step 1: case: item<=ptr → data
  5. Step 1: ptr1 = ptr
  6. Step 2: ptr=ptr→ lc
  7. Step 2: case: item>ptr → data
  8. Step 1: ptr1=ptr
  9. Step 2: ptr = ptr→ rc
  10. Step 3: endCase
  11. Step 4: endWhile
  12. Step 5: if(ptr==NULL) then
  13. Step 1: new = getNode(node)
  14. Step 2: new → data = item
  15. Step 3: new→ rc = new→lc = NULL
  16. Step 4: if(ptr→ dara <= item) then
  17. Step 1: ptr1→ rc = new
  18. Step 5: else
  19. Step 1: ptr1→lc = new
  20. Step 6: endIf
  21. Step 6: endif
  22. Step 7: Stop

**Algorithm for Sorting**

**Input:** Root node of the binary tree containing the elements to be sorted and a array in which elements are to be inserted in sorted order

**Output :** All the elements sorted

**Data Structure used:** Binary Search trees, array

Steps

1. Step 1: Start // i is initialized to zero
2. Step 2: if(root!=NULL) then
3. Step 1: bst\_sort(root→lc,arr)
4. Step 2: arr[i] = root→ value
5. Step 3: i++
6. Step 4: bst\_sort(root→ rc,arr)
7. Step 3: else
8. Step 1: return
9. Step 4: endif
10. Step 5: Stop

**Program Code**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Sorting using binary search tree

\* Done By Rohit Karunakaran

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct binary\_search\_tree\_node{

struct binary\_search\_tree\_node\* lc;

struct binary\_search\_tree\_node\* rc;

int value;

}node;

void insert\_node(node\*\* root,int value){

int flag=1;

node\* ptr=\*root;

if(ptr!=NULL){

while(ptr!=NULL&&flag){

if(ptr->value<value){

if(ptr->rc==NULL){

ptr->rc = (node\*)malloc(sizeof(node));

ptr->rc->lc = ptr->rc->rc =NULL;

ptr->rc->value = value;

flag=0;

}

else{

ptr= ptr->rc;

}

}

else{

if(ptr->lc==NULL){

ptr->lc = (node\*)malloc(sizeof(node));

ptr->lc->lc = ptr->lc->rc =NULL;

ptr->lc->value = value;

flag=0;

}

else{

ptr = ptr->lc;

}

}

}

}

else{

//Root is empty

\*root = (node\*)malloc(sizeof(node));

(\*root) ->lc = (\*root)->rc = NULL;

(\*root)->value = value;

}

}

int index =0;

void bstSort(node\* root,int arr[]){

if(root!=NULL){

bstSort(root->lc,arr);

arr[index] = root->value; index++;

bstSort(root->rc,arr);

}

else{

return;

}

}

int main(){

node\* root = NULL;

int n;

printf("Enter the number of elements to be sorted :");

scanf("%d",&n);

int arr[n];

printf("Enter the elements in the array : ");

for(int i=0;i<n;i++){

int elem;

scanf("%d",&elem);

insert\_node(&root,elem);

}

bstSort(root,arr);

printf("The Sorted array of elemets are: ");

for(int i=0;i<n;i++){

printf("%d ",arr[i]);

}

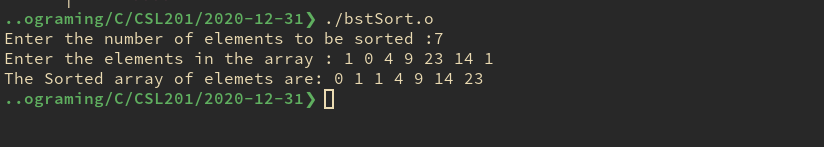
printf("\n");

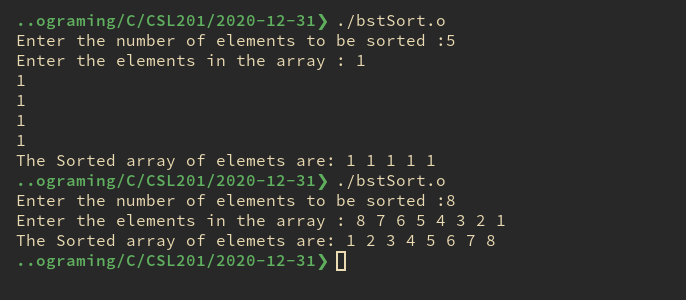
return 0;

}

**Result:** The program compiled successfully and required output was obtained

**Sample input and output**

****

****