# Data Structures Through C

# Home Assignment

Name: Kandikonda S V S K Devi Prakash

Roll no: 198W1A0294

Sec: EEE-B (3/4B.Tech)

Insertion of Binary Search Tree:

### Code:

// C program to demonstrate insert

// operation in binary

// search tree.

#include <stdio.h>

#include <stdlib.h>

struct node {

int key;

struct node \*left, \*right;

};

// A utility function to create a new BST node

struct node\* newNode(int item)

{

struct node\* temp

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

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

// A utility function to do inorder traversal of BST

void inorder(struct node\* root)

{

if (root != NULL) {

inorder(root->left);

printf("%d \n", root->key);

inorder(root->right);

}

}

/\* A utility function to insert

a new node with given key in

\* BST \*/

struct node\* insert(struct node\* node, int key)

{

/\* If the tree is empty, return a new node \*/

if (node == NULL)

return newNode(key);

/\* Otherwise, recur down the tree \*/

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

/\* return the (unchanged) node pointer \*/

return node;

}

// Driver Code

int main()

{

/\* Let us create following BST

50

/ \

30 70

/ \ / \

20 40 60 80 \*/

struct node\* root = NULL;

root = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

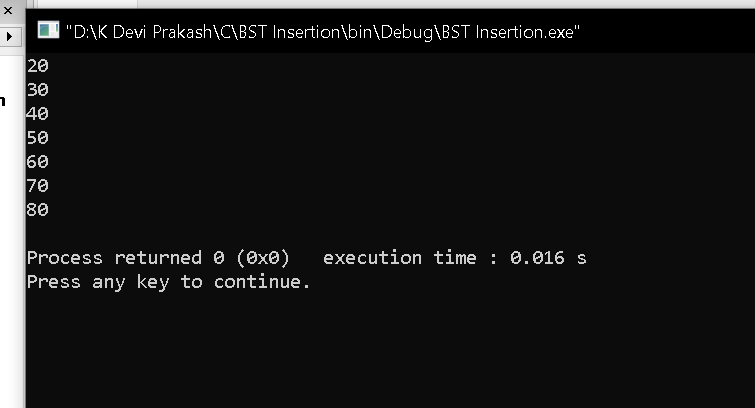
// print inoder traversal of the BST

inorder(root);

return 0;

}

Output:



Binary Search Tree deletion:

Code:

// C program to demonstrate

// delete operation in binary

// search tree

#include <stdio.h>

#include <stdlib.h>

struct node {

int key;

struct node \*left, \*right;

};

// A utility function to create a new BST node

struct node\* newNode(int item)

{

struct node\* temp

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

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

// A utility function to do inorder traversal of BST

void inorder(struct node\* root)

{

if (root != NULL) {

inorder(root->left);

printf("%d ", root->key);

inorder(root->right);

}

}

/\* A utility function to

insert a new node with given key in

\* BST \*/

struct node\* insert(struct node\* node, int key)

{

/\* If the tree is empty, return a new node \*/

if (node == NULL)

return newNode(key);

/\* Otherwise, recur down the tree \*/

if (key < node->key)

node->left = insert(node->left, key);

else

node->right = insert(node->right, key);

/\* return the (unchanged) node pointer \*/

return node;

}

/\* Given a non-empty binary search

tree, return the node

with minimum key value found in

that tree. Note that the

entire tree does not need to be searched. \*/

struct node\* minValueNode(struct node\* node)

{

struct node\* current = node;

/\* loop down to find the leftmost leaf \*/

while (current && current->left != NULL)

current = current->left;

return current;

}

/\* Given a binary search tree

and a key, this function

deletes the key and

returns the new root \*/

struct node\* deleteNode(struct node\* root, int key)

{

// base case

if (root == NULL)

return root;

// If the key to be deleted

// is smaller than the root's

// key, then it lies in left subtree

if (key < root->key)

root->left = deleteNode(root->left, key);

// If the key to be deleted

// is greater than the root's

// key, then it lies in right subtree

else if (key > root->key)

root->right = deleteNode(root->right, key);

// if key is same as root's key,

// then This is the node

// to be deleted

else {

// node with only one child or no child

if (root->left == NULL) {

struct node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL) {

struct node\* temp = root->left;

free(root);

return temp;

}

// node with two children:

// Get the inorder successor

// (smallest in the right subtree)

struct node\* temp = minValueNode(root->right);

// Copy the inorder

// successor's content to this node

root->key = temp->key;

// Delete the inorder successor

root->right = deleteNode(root->right, temp->key);

}

return root;

}

// Driver Code

int main()

{

/\* Let us create following BST

50

/ \

30 70

/ \ / \

20 40 60 80 \*/

struct node\* root = NULL;

root = insert(root, 50);

root = insert(root, 30);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 70);

root = insert(root, 60);

root = insert(root, 80);

printf("Inorder traversal of the given tree \n");

inorder(root);

printf("\nDelete 20\n");

root = deleteNode(root, 20);

printf("Inorder traversal of the modified tree \n");

inorder(root);

printf("\nDelete 30\n");

root = deleteNode(root, 30);

printf("Inorder traversal of the modified tree \n");

inorder(root);

printf("\nDelete 50\n");

root = deleteNode(root, 50);

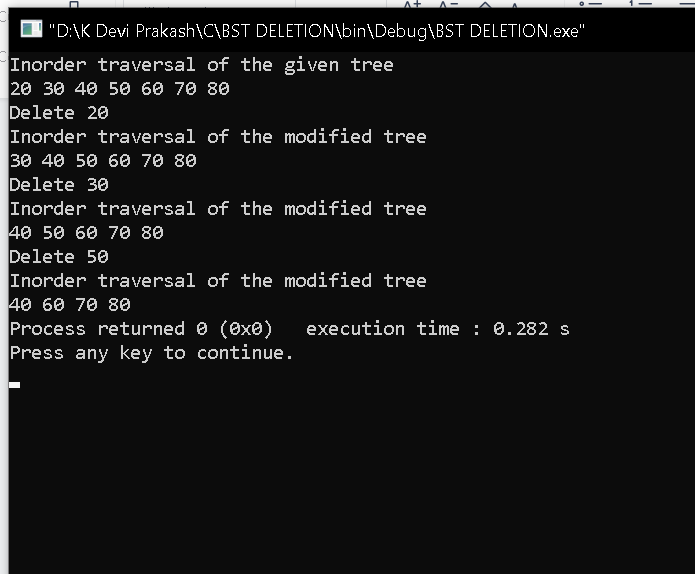
printf("Inorder traversal of the modified tree \n");

inorder(root);

return 0;

}

Output:



Searching:

Code

#include <stdio.h>

#include <stdlib.h>

// Basic struct of Tree

struct node

{

int val;

struct node \*left, \*right;

};

// Function to create a new Node

struct node\* newNode(int item)

{

struct node\* temp = (struct node \*)malloc(sizeof(struct node));

temp->val = item;

temp->left = temp->right = NULL;

return temp;

}

// A sample C function to check if a given node exists in a binary search tree or not

int search(struct node\* root, int value)

{

// while is used to traverse till the end of tree

while (root != NULL){

// checking condition and passing right subtree & recusing

if (value > root->val)

root = root->right;

// checking condition and passing left subtree & recusing

else if (value < root->val)

root = root->left;

else

return 1; // if the value is found return 1

}

return 0;

}

int main()

{

/\*Root is created first\*/

struct node\* root = newNode(21);

root->left = newNode(16);

root->right = newNode(25);

root->left->left = newNode(10);

root->left->right = newNode(18);

root->right->left = newNode(22);

root->right->right = newNode(28);

root->left->left->left = newNode(8);

root->left->left->right = newNode(12);

int item = 10;

// Function to find item in the tree

int found = search(root,item);

if(found)

printf("%d value is found in the tree",item);

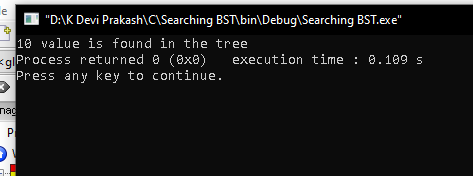
else

printf("%d value not found",item);

return 0;

}

Output:



Insertion sort:

Code:

#include<stdio.h>

void printArray(int\* A, int n){

int i;

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

{

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

}

printf("\n");

}

void insertionSort(int \*A, int n){

int key, j,i;

// Loop for passes

for ( i = 1; i <= n-1; i++)

{

key = A[i];

j = i-1;

// Loop for each pass

while(j>=0 && A[j] > key){

A[j+1] = A[j];

j--;

}

A[j+1] = key;

}

}

int main(){

// -1 0 1 2 3 4 5

// 12,| 54, 65, 07, 23, 09 --> i=1, key=54, j=0

// 12,| 54, 65, 07, 23, 09 --> 1st pass done (i=1)!

// 12, 54,| 65, 07, 23, 09 --> i=2, key=65, j=1

// 12, 54,| 65, 07, 23, 09 --> 2nd pass done (i=2)!

// 12, 54, 65,| 07, 23, 09 --> i=3, key=7, j=2

// 12, 54, 65,| 65, 23, 09 --> i=3, key=7, j=1

// 12, 54, 54,| 65, 23, 09 --> i=3, key=7, j=0

// 12, 12, 54,| 65, 23, 09 --> i=3, key=7, j=-1

// 07, 12, 54,| 65, 23, 09 --> i=3, key=7, j=-1--> 3rd pass done (i=3)!

// Fast forwarding and 4th and 5th pass will give:

// 07, 12, 54, 65,| 23, 09 --> i=4, key=23, j=3

// 07, 12, 23, 54,| 65, 09 --> After the 4th pass

// 07, 12, 23, 54, 65,| 09 --> i=5, key=09, j=4

// 07, 09, 12, 23, 54, 65| --> After the 5th pass

int A[] = {12, 54, 65, 7, 23, 9};

int n = 6;

printArray(A, n);

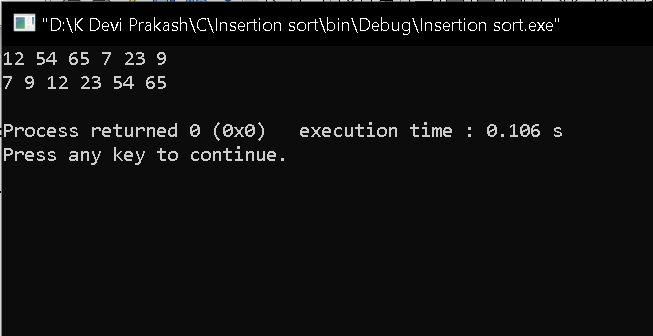
insertionSort(A, n);

printArray(A, n);

return 0;

}

Output:



Bubble sort:

Code:

#include<stdio.h>

void printArray(int\* A, int n){

int i;

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

{

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

}

printf("\n");

}

void bubbleSort(int \*A, int n){

int j;

int temp;

int isSorted = 0;

int i;

for ( i = 0; i < n-1; i++) // For number of pass

{

printf("Working on pass number %d\n", i+1);

for (j = 0; j <n-1-i ; j++) // For comparison in each pass

{

if(A[j]>A[j+1]){

temp = A[j];

A[j] = A[j+1];

A[j+1] = temp;

}

}

}

}

void bubbleSortAdaptive(int \*A, int n){

int temp,i,j;

int isSorted = 0;

for ( i = 0; i < n-1; i++) // For number of pass

{

printf("Working on pass number %d\n", i+1);

isSorted = 1;

for (j = 0; j <n-1-i ; j++) // For comparison in each pass

{

if(A[j]>A[j+1]){

temp = A[j];

A[j] = A[j+1];

A[j+1] = temp;

isSorted = 0;

}

}

if(isSorted){

return;

}

}

}

int main(){

// int A[] = {12, 54, 65, 7, 23, 9};

int A[] = {1, 2, 5, 6, 12, 54, 625, 7, 23, 9, 987};

// int A[] = {1, 2, 3, 4, 5, 6};

int n = 11;

printArray(A, n); // Printing the array before sorting

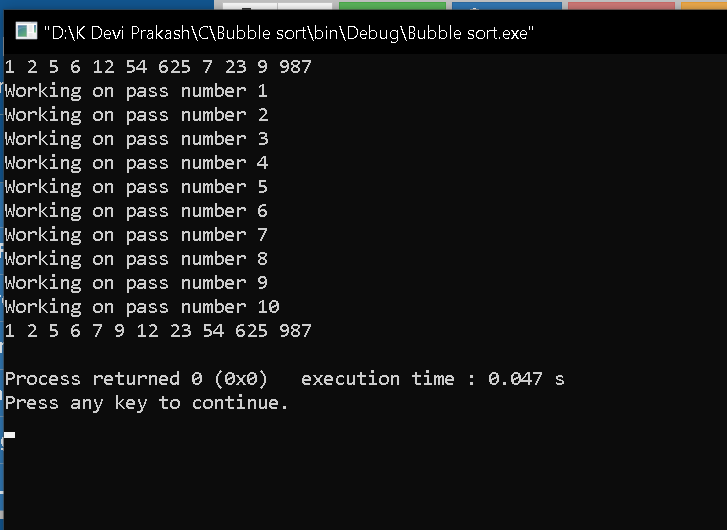
bubbleSort(A, n); // Function to sort the array

printArray(A, n); // Printing the array before sorting

return 0;

}

Output:



Selection Sort:

Code:

#include<stdio.h>

void printArray(int\* A, int n){

int i;

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

{

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

}

printf("\n");

}

void selectionSort(int \*A, int n){

int indexOfMin, temp,i,j;

printf("Running Selection sort...\n");

for (i = 0; i < n-1; i++)

{

indexOfMin = i;

for ( j = i+1; j < n; j++)

{

if(A[j] < A[indexOfMin]){

indexOfMin = j;

}

}

// Swap A[i] and A[indexOfMin]

temp = A[i];

A[i] = A[indexOfMin];

A[indexOfMin] = temp;

}

}

int main(){

// Input Array (There will be total n-1 passes. 5-1 = 4 in this case!)

// 00 01 02 03 04

// |03, 05, 02, 13, 12

// After first pass

// 00 01 02 03 04

// 02,|05, 03, 13, 12

// After second pass

// 00 01 02 03 04

// 02, 03,|05, 13, 12

// After third pass

// 00 01 02 03 04

// 02, 03, 05,|13, 12

// After fourth pass

// 00 01 02 03 04

// 02, 03, 05, 12,|13

int A[] = {3, 5, 2, 13, 12};

int n = 5;

printArray(A, n);

selectionSort(A, n);

printArray(A, n);

return 0;

}

Output:

