**QUESTION: 1**

Given pseudo code for a linear time algorithm that verifies that AVL tree is correctly

maintained. Assume every node has field key, data, height, left and right and that key

can be compared with <,==, and >. The algorithm should also verify following.

 The tree is a binary search tree.

 The height information stored in each node is correct.

 Every node is balanced.

Your code should have a Boolean return type and return true if the tree is a valid AVL

Tree or false if it violates any of the above properties

Ex.No:2.4(a)

AVL TREE BALANCED

**AIM:**

To write a program verifies that AVL tree is correctly maintained. Assume every node has field key, data, height, left and right and that key.

**PSEUDOCODE:**

Algorithm to verify AVL tree is correctly maintained.

//Returns the inorder traversal of tree and verification result.

//Input: elements of tree.

//Output: Inorder traversal of tree and verification result.

bool isBalanced(struct Node\* root)

{

Initialize int lh,rh;

if (root == NULL) Do

return true;

lh = height(root->left);

rh = height(root->right);

if (abs(lh - rh) <= 1 && isBalanced(root->left) && isBalanced(root->right)) Do

return true;

return false;

}

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct Node

{

int key;

struct Node \*left;

struct Node \*right;

int height;

};

int height(struct Node \*N)

{

if (N == NULL)

return 0;

return N->height;

}

int max(int a, int b)

{

return (a > b)? a : b;

}

struct Node\* newNode(int key)

{

struct Node\* node = (struct Node\*)

malloc(sizeof(struct Node));

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1;

return(node);

}

struct Node \*rightRotate(struct Node \*k1){

struct Node \*k2;

k2=k1->left;

k1->left=k2->right;

k2->right=k1;

k1->height=max(height(k1->left),height(k1->right))+1;

k2->height=max(height(k2->left),k1->height)+1;

return k2;

}

bool isBalanced(struct Node\* root)

{

int lh;

int rh;

if (root == NULL)

return true;

lh = height(root->left);

rh = height(root->right);

if (abs(lh - rh) <= 1 && isBalanced(root->left) && isBalanced(root->right))

return true;

return false;

}

struct Node \*leftRotate(struct Node \*k1){

struct Node \*k2;

k2=k1->right;

k1->right=k2->left;

k2->left=k1;

k1->height=max(height(k1->left),height(k1->right))+1;

k2->height=max(height(k2->left),k1->height)+1;

return k2;

}

int getBalance(struct Node \*N)

{

if (N == NULL)

return 0;

return height(N->left) - height(N->right);

}

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

{

if (node == NULL)

return(newNode(key));

if (key < node->key)

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

else if (key > node->key)

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

else

return node;

node->height = 1 + max(height(node->left),height(node->right));

int balance = getBalance(node);

if (balance > 1 && key < node->left->key)

return rightRotate(node);

if (balance < -1 && key > node->right->key)

return leftRotate(node);

if (balance > 1 && key > node->left->key)

{

node->left = leftRotate(node->left);

return rightRotate(node);

}

if (balance < -1 && key < node->right->key)

{

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

void inorder(struct Node \*temp)

{

if(temp!=NULL)

{

inorder(temp->left);

printf("%d\t",temp->key);

inorder(temp->right);

}

}

int main()

{

struct Node \*root = NULL;

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 30);

root = insert(root, 40);

root = insert(root, 50);

root = insert(root, 25);

printf("Inorder traversal of the constructed AVL"

" tree is \n");

inorder(root);

if(isBalanced(root))

printf("\nAVL TREE IS PERFORMING IN CORRECT WAY\n");

else

printf("AVL TREE IS NOT PERFORMING IN CORRECT WAY\n");

return 0;

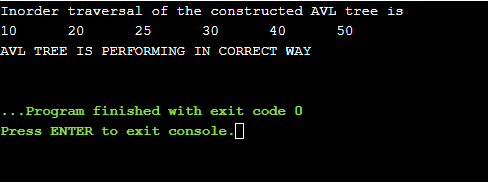
}

**DATA STRUCTURE USED:** TREE

**ROUTINE:** RECURSIVE

**TIME COMPLEXITY:** O(N)

**OUTPUT:**

****

**RESULT:**

Thus the program to write a program verifies that AVL tree is correctly maintained. Assume every node has field key, data, height, left and right and that key.

**QUESTION 2:**

Given an unsorted array. The array has this property that every element in array is at most

k distance from its position in sorted array where k is a positive integer smaller than size

of array. Implement the sorting algorithm can be easily modified for sorting this array.

Ex.No:2.4(b)

**HEAP SORT**

**AIM:**

To write a program for heap sorting algorithm.

**PSEUDOCODE:**

Algorithm to Implement Heap Sort.

//Returns the sorted array.

//Input: elements of tree.

//Output: Sort the array using heap sort.

void heapify(int a[], int n)

{

int parent,p\_value,child,item,i,flag;

for(i=n/2;i>=1;i--) do

parent=i;

p\_value=a[parent];

flag=0;

while((!flag)&&(2\*parent<=n)) do

{ child=2\*parent;

if(child<n)do

if(a[child]<a[child+1])do

child=child+1;

if(p\_value>=a[child])do

flag=1;

else

{

a[parent]=a[child];

parent=child;

a[parent]=p\_value;

}

}

}

}

void heapSort(int a[], int n)

{

int i,t;

heapify(a, n);

for (i=n; i>=1; i--) do

{ t = a[1];

a[1]= a[i];

a[i] = t;

heapify(a, i-1);

} }

**SOURCE CODE:**

#include<stdio.h>

void heapify(int a[], int n)

{

int parent,p\_value,child,item,i,flag;

for(i=n/2;i>=1;i--)

{

parent=i;

p\_value=a[parent];

flag=0;

while((!flag)&&(2\*parent<=n))

{

child=2\*parent;

if(child<n)

if(a[child]<a[child+1])

child=child+1;

if(p\_value>=a[child])

flag=1;

else

{

a[parent]=a[child];

parent=child;

a[parent]=p\_value;

}

}

}

}

void heapSort(int a[], int n)

{

int i,t;

heapify(a, n);

for (i=n; i>=1; i--)

{

t = a[1];

a[1]= a[i];

a[i] = t;

heapify(a, i-1);

}

}

void main()

{ int n,i;

printf("Enter array size:");

scanf("%d",&n);

int a[n];

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

scanf("%d",&a[i]);

heapSort(a,n);

printf("printing sorted elements\n");

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

printf("%d\t",a[i]);

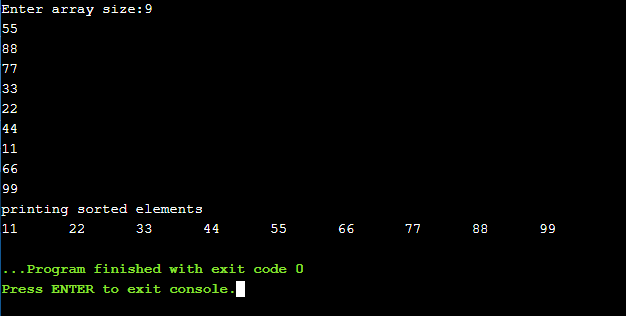
}

**DATA STRUCTURE USED:** TREE

**ROUTINE:** RECURSIVE

**TIME COMPLEXITY:** O(N LOGN)

**OUTPUT:**

****

**RESULT:**

Thus the program to implement the heap sort is done successfully.