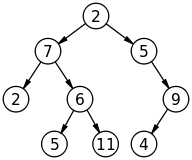
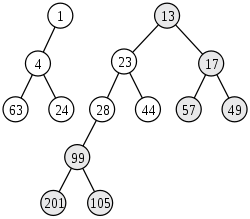
**1)(a)**

(i)

In computer science, a **binary tree** is a **tree** data structure in which each node has at most two children, which are referred to as the left child and the right child.



(ii) A **skew** heap (or self-adjusting heap) is a heap data structure implemented as a **binary tree**.**Skew** heaps are advantageous because of their ability to merge more quickly than **binary**heaps.



(iii) A **full binary tree** (sometimes proper **binary tree** or 2-**tree**) is a **tree** in which every node other than the leaves has two children. A**complete binary tree** is a **binary tree** in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible

x

/ \

/ \

x x

/ \ / \

x x x x

/ \ /

x x x

(iv)

If every non-leaf node in a **binary tree** has nonempty left and right subtrees, the **tree** is termed a **strictly binary tree**. Or, to put it another way, all of the nodes in a **strictly binary tree** are of degree zero or two, never degree one.

x

/ \

/ \

x x

/ \

x x

/ \

x x

**1(b)**

#include<iostream.h>

#include<conio.h>

#include<process.h>

#include<stdlib.h>

#include<stdio.h>

//using namespace std;

class node{ int data;

node \*next, \*prev;

node \*tail,\*tail1,\*tail2;

int count;

public:

node()

{

//header=NULL;

// head = NULL;

tail=tail1=tail2 = NULL;

count=0;

}

void insert();

void disp(node\*);

void meth();

void display(node\*);

};node \*h1,\*h2,\*head;

void main()

{node obj;

int n;

clrscr();

cout<<"enter no. of elements:-"<<endl;

cin>>n;

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

obj.insert();

obj.disp(head);

obj.meth();

obj.display(h1);

obj.display(h2);

cout<<endl;

getch();

// return 0;

}

// return 0;

void node::insert()

{

node \*temp = new node;

temp->next = NULL;

cout<<"enter value"<<endl;

cin>>temp->data;

if(head!=NULL && tail!=NULL)

{tail->next=temp;

temp->prev=tail;

temp->next=head;

head->prev=temp;

tail=temp;

count++;

head->data=count;

}

else

{

head=new node();

tail=temp;

head->next=temp;

temp->prev=head;

temp->next=head;

head->prev=temp;

count++;

head->data=count;

}

// cout<<tail->next->data<<"\n";

}

void node::disp(node\* head)

{

node \*temp=head;

if(count==0)

{

cout<<"empty!"<<endl;

return;

}

//cout<<temp->data<<"\t";

temp=temp->next;

while(temp!=head)

{cout<<temp->data<<"\t";

temp=temp->next;

}

cout<<endl;

}

void node::display(node\* head)

{

node \*temp=head;

if(temp==NULL)

{

cout<<"empty!"<<endl;

return;

}

//cout<<temp->data<<"\t";

while(temp!=NULL)

{cout<<temp->data<<"\t";

temp=temp->next;

}

cout<<endl;

}

void node::meth()

{

node\* t=new node;

t=head->next;

int coun=2;

while(t!=head)

{

node \*m=t;

if(coun%2==0)

{

if(h1==NULL)

{

h1=tail1=m;

h1->next=NULL;

}

else{

tail1->next=m;

tail1=m;

tail1->next=NULL;

}

}

else{

if(h2==NULL)

{

h2=tail2=m;

tail2->next=NULL;

}

else{

tail2->next=m;

tail2=m;

tail2->next=NULL;

}

}

t=t->next;

coun++;

}

}

**2(a)**

#include<stdio.h>

#include<stdlib.h>

#include<iostream.h>

#include<conio.h>

//using namespace std;

//Structure of a node

class node

{

public:

int info;

node \*left;

node \*right;

node(){

left=NULL;

right=NULL;

}

node\* createNode(node\*,int);

void inorder(node\*);

};

//Function to add a node to tree

node\* node::createNode(node\* root, int item)

{

if(root == NULL)

{

node\* temp = new node;;

temp -> info = item;

temp -> left = NULL;

temp -> right = NULL;

root = temp;

}

else if((item) <= (root->info))

root->left = createNode((root->left), item);

else if((item)> (root->info))

root -> right = createNode((root->right), item);

return root;

}

//Inorder display function

void node::inorder(node\* root)

{

if(root != NULL)

{

inorder(root->left);

cout<<root->info<<"\t";

inorder(root->right);

}

}

void main()

{

clrscr();

node obj;

int choice, item;

node\* root = NULL;

while(1)

{

cout<<"\n1.Add Element\n2.Display\n3.Exit\n";

cin>>choice;

switch(choice)

{

case 1:

cout<<"\nEnter element to be added\n";

cin>>item;

root=obj.createNode(root, item);

break;

case 2:

cout<<"\nINORDER DISPLAY\n";

obj.inorder(root);

break;

case 3:

exit(0);

}

}

getch();

}

**2(b)**

#include<stdio.h>

#include<stdlib.h>

#include<iostream.h>

#include<conio.h>

//using namespace std;

//Structure of a node

class node

{

public:

int info;

node \*left;

node \*right;

node(){

left=NULL;

right=NULL;

}

node\* createNode(node\*,int);

int largest(node\*);

};

//Function to add a node to tree

node\* node::createNode(node\* root, int item)

{

if(root == NULL)

{

node\* temp = new node;;

temp -> info = item;

temp -> left = NULL;

temp -> right = NULL;

root = temp;

}

else if((item) <= (root->info))

root->left = createNode((root->left), item);

else if((item)> (root->info))

root -> right = createNode((root->right), item);

return root;

}

int node::largest( node\* root )

{

if ( root == NULL )

return -1;

if(root->right==NULL)

return root->info;

largest(root->right);

/\*

For a general tree

int left = largest(root->left);

int right = largest ( root->right);

if( root->info > left && root->info > right )

return root->info;

else

return max ( left, right );\*/

}

void main()

{

clrscr();

node obj;

int choice, item, a;

node\* root = NULL;

while(1)

{

cout<<"\n1.Add Element\n2.Display Largest\n3.Exit\n";

cin>>choice;

switch(choice)

{

case 1:

cout<<"\nEnter element to be added\n";

cin>>item;

root=obj.createNode(root, item);

break;

case 2:

cout<<"\nLargest Number\n";

a=obj.largest(root);

cout<<a;

break;

case 3:

exit(0);

}

}

getch();

//return 0;

}

**3)**

//All parts in one program.

#include<stdio.h>

#include<stdlib.h>

#include<iostream.h>

#include<conio.h>

//using namespace std;

//Structure of a node

class node

{

public:

int info;

node \*left;

node \*right;

node(){

left=NULL;

right=NULL;

}

node\* createNode(node\*,int);

void inorder(node\*);

void swapp(node\*);

node\* copyy(node\*);

bool tree\_compare(node\*,node\*);

int getLeafCount(node\*);

int countt(node\*);

};

void node::inorder(node\* root)

{

if(root != NULL)

{

inorder(root->left);

cout<<root->info<<"\t";

inorder(root->right);

}

}

int node::countt(node \*n)

{

int c = 1;

if (n == NULL)

return 0;

else

{

c += countt(n->left);

c += countt(n->right);

return c;

}

}

int node::getLeafCount(node\* node)

{

if(node == NULL)

return 0;

if(node->left == NULL && node->right==NULL)

return 1;

else

return getLeafCount(node->left)+

getLeafCount(node->right);

}

/\*This mthod checks the equality of orders of data too.

If you just want to check the equality of data, inorder traversal order of both the tree will be same\*/

bool node::tree\_compare (node\* t1, node\* t2)

{

if (t1 == t2) return true;

if ((t1 == NULL) || (t2 == NULL)) return false;

return ((t1->info == t2->info) && tree\_compare (t1->left, t2->left )

&& tree\_compare (t1->right, t2->right));

}

void node::swapp(node\* root)

{

if(root==NULL)

return;

if(root->right==0 && root->left==0)

return ;

else

{

node\* temp=root->left;

root->left=root->right;

root->right=temp;

swapp(root->left);

swapp(root->right);

}

}

//Function to add a node to tree

node\* node::createNode(node\* root, int item)

{

if(root == NULL)

{

node\* temp = new node;;

temp -> info = item;

temp -> left = NULL;

temp -> right = NULL;

root = temp;

}

else if((item) <= (root->info))

root->left = createNode((root->left), item);

else if((item)> (root->info))

root -> right = createNode((root->right), item);

return root;

}

node\* node::copyy(node \*root) {

node \*new\_root;

if(root!=NULL){

new\_root=new node;

new\_root->info=root->info;

new\_root->left=copyy(root->left);

new\_root->right=copyy(root->right);

} else return NULL;

return new\_root;

}

void main()

{

clrscr();

node obj;

int choice, item, a;

node\* root = NULL;

node\* root1=NULL;

node\* nr;

bool b;

int t,y;

while(1)

{

cout<<"\n1.Add Element\n2.Swap\n3.Copy\n4.Display\n5.Add elements to the second tree\n6.Test\n7.Count\n8.Exit";

cin>>choice;

switch(choice)

{

case 1:

cout<<"\nEnter element to be added\n";

cin>>item;

root=obj.createNode(root, item);

//cout<<"Inorder Display\n";

//obj.inorder(root);

break;

case 2:

cout<<"\nAfter swapping node\n";

obj.swapp(root);

obj.inorder(root);

break;

case 3:

cout<<"\nCopied tree is\n";

nr=obj.copyy(root);

obj.inorder(nr);

break;

case 4:

obj.inorder(root);

break;

case 5:

cout<<"Enter the values of the second tree";

cout<<"\nEnter element to be added\n";

cin>>item;

root1=obj.createNode(root1, item);

cout<<"Inorder Display\n";

obj.inorder(root1);

break;

case 6:

b=obj.tree\_compare(root,root1);

if(b==true)

cout<<"Trees are equal.";

else

cout<<"Tress are not equal.";

break;

case 7:

t=obj.countt(root);

y=obj.getLeafCount(root);

cout<<"Number of non-leaf node is "<<t-y;

break;

case 8:

exit(0);

}

}

getch();

//return 0;

}

**4(a)**

#include<iostream.h>

#include<conio.h>

#include<process.h>

class node{ int data;

node \*next, \*prev;

node \*head, \*tail;

public:

node()

{head = NULL;

tail = NULL;

}

void insert();

void disp();

void rev();

};

void main()

{

node obj;

int ch,n;

clrscr();

cout<<"enter no. of elements:-"<<endl;

cin>>n;

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

obj.insert();

obj.disp();

cout<<endl;

cout<<"After reversing\n";

obj.rev();

getch();

}

void node::insert()

{node \*temp = new node;

temp->next = NULL;

temp->prev = NULL;

cout<<"enter value"<<endl;

cin>>temp->data;

if(head!=NULL && tail!=NULL)

{tail->next=temp;

temp->prev=tail;

tail=temp;

}

else

tail=head=temp;

}

void node::disp()

{node \*temp=head;

if(temp==NULL)

cout<<"empty!"<<endl;

while(temp!=NULL)

{cout<<temp->data<<"\t";

temp=temp->next;

}

cout<<endl;

}

void node::rev()

{node \*temp=head, \*r;

head=NULL;

while(temp!=NULL)

{

r=temp->next;

temp->prev=temp->next;

temp->next=head;

head=temp;

temp=r;

}

}

**4(b)**

#include<iostream.h>

#include<conio.h>

#include<process.h>

#include<stdlib.h>

#include<stdio.h>

//using namespace std;

class node{ int data;

node \*next, \*prev;

node \*head, \*tail;

int count;

public:

node()

{

//header=NULL;

// head = NULL;

head=tail= NULL;

count=0;

}

void insert();

void disp();

void dupre();

};

void main()

{node obj;

int n;

clrscr();

cout<<"enter no. of elements:-"<<endl;

cin>>n;

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

obj.insert();

obj.disp();

obj.dupre();

cout<<"After removing the duplicate elements\n";

obj.disp();

getch();

//return 0;

}

// return 0;

void node::insert()

{

node \*temp = new node;

temp->next = NULL;

cout<<"enter value"<<endl;

cin>>temp->data;

if(head!=NULL && tail!=NULL)

{tail->next=temp;

temp->prev=tail;

//temp->next=head;

//head->prev=temp;

tail=temp;

count++;

head->data=count;

}

else

{

head=new node();

tail=temp;

head->next=temp;

temp->prev=head;

count++;

head->data=count;

}

// cout<<tail->next->data<<"\n";

}

void node::disp()

{

node \*temp=head;

if(count==0)

{

cout<<"empty!"<<endl;

return;

}

//cout<<temp->data<<"\t";

temp=temp->next;

while(temp!=NULL)

{cout<<temp->data<<"\t";

temp=temp->next;

}

cout<<endl;

}

void node::dupre()

{

node \*temp=head;

temp=temp->next;

while(temp!=NULL)

{

if(temp->data==temp->next->data)

{

node\*t=temp;

while(t->next->data==t->data)

t=t->next;

t=t->next;

temp->next=t;

t->prev=temp;

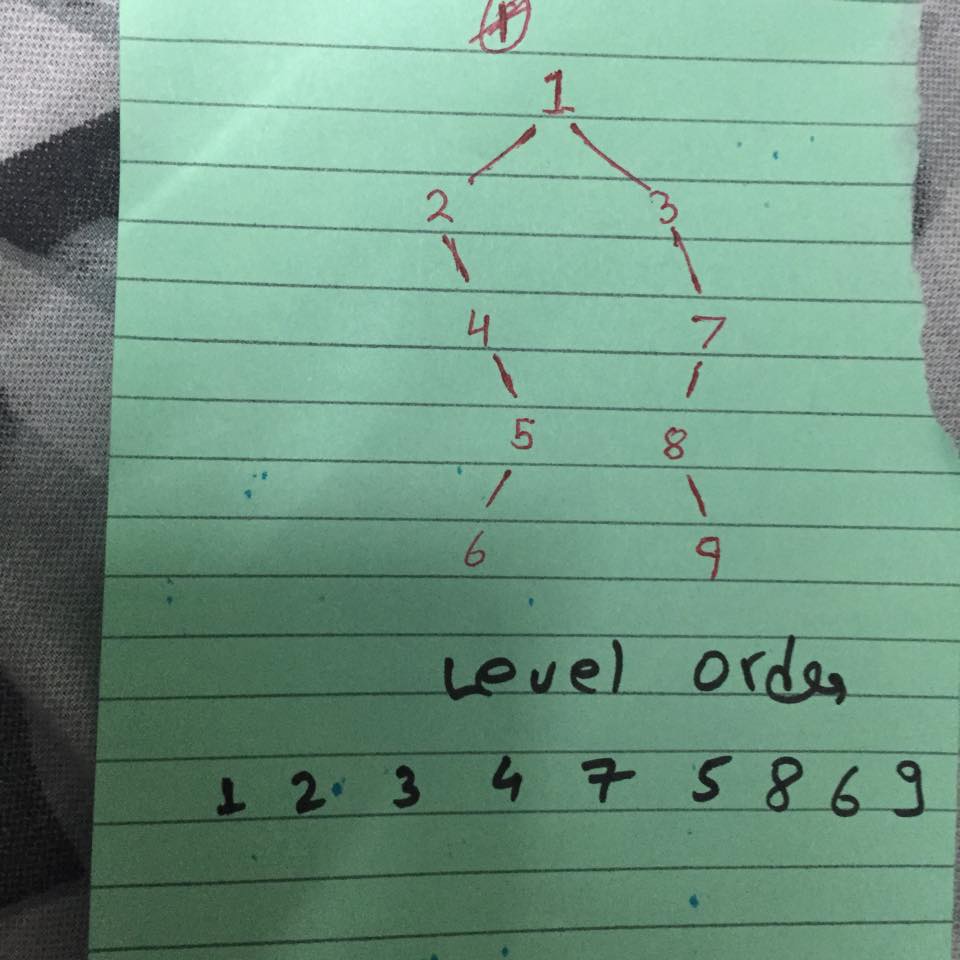
}

temp=temp->next;

}

}

**5(a)**



//I am also uploading the level order traversal program which I am not sure is necessary to write in this question.

#include<stdio.h>

#include<stdlib.h>

#include<iostream.h>

class node

{

int info;

node \*left;

node \*right;

node \*array[10];

int front;

int rear;

void deque();

void enque(node\*);

node\* createNode(node\*,int);

void levelorder(node\*);

};node Q;

//Enqueue function

void node::enqueue(node\* x)

{

if((Q.front + 1)%10 == (Q.rear))

cout<<"\nOVERFLOW!!\n";

else

{

Q.array[Q.front] = x;

Q.front = (Q.front + 1)%10;

}

}

//Dequeue function

void node::dequeue()

{

if(Q.rear == Q.front)

cout<<"\nUNDERFLOW\n";

else

Q.rear = (Q.rear+1)%10;

}

//Function to add a node to tree

node\* node::createNode(node\* root, int item)

{

if(root == NULL)

{

node\* temp;

temp = new node;

temp -> info = item;

temp -> left = NULL;

temp -> right = NULL;

root = temp;

}

else if((item) <= (root->info))

root->left = createNode((root->left), item);

else

root -> right = createNode((root->right), item);

return root;

}

void node::levelorder(node\* root)

{

if(root == NULL)

cout<<"\nTREE IS EMPTY\n";

else

{

enqueue(root);

while(Q.front != Q.rear)

{

node\* current = Q.array[Q.rear];

cout<<current->info;

if(current->left != NULL)

enqueue(current->left);

if(current->right != NULL)

enqueue(current->right);

dequeue();

}

}

}

//Main Function

void main()

{

int choice, item;

node obj;

node\* root = NULL;

Q.front=0;

Q.rear=0;

clrscr();

while(1)

{

cout<<"\n1.Add Element\n2.Display\n3.Exit\n");

cin>>choice;

switch(choice)

{

case 1:

cout<<"\nEnter element to be added\n";

cin>>item;

root=obj.createNode(root, item);

break;

case 2:

cout<<"\nLEVEL OREDER DISPLAY\n";

obj.levelorder(root);

break;

case 3:

exit(0);

}

getch();

}

}

**5(b)**

//If the inorder display is sorted it means the tree is BST

//ar and i are global.

void node::inorder(node\* root)

{

if(root != NULL)

{

inorder(root->left);

ar[i]=root->info;

i++;

inorder(root->right);

}

}

//In main

for(int j=0;j<i-1;j++)

{

if(ar[i]>ar[i+1])

{

cout<<"Tree is not a BST";

flag=1;

break;

}

}

if(flag==0)

cout<<"Tree is BST";