**Question 1:**

**Code:**

**#include<iostream>**

**#include<assert.h>**

**using namespace std;**

**class MaxHeap{**

**private:**

**struct node{**

**int key;**

**string msg;**

**int date;**

**};**

**node \*arr;**

**int capacity;**

**public:**

**int totalItems;**

**void doubleCapacity(){**

**if(this->arr == NULL){**

**this->arr = new node[1];**

**this->capacity = 1;**

**return;**

**}**

**int newCapacity = capacity \* 2;**

**node \*newArr = new node[newCapacity];**

**for(int i = 0; i < this->totalItems; i++){**

**newArr[i] = this->arr[i];**

**}**

**delete this->arr;**

**this->capacity = newCapacity;**

**this->arr = newArr;**

**}**

**void shiftUp(int index){**

**if(index < 1){**

**return;**

**}**

**int parent = (index-1)/2;**

**if(this->arr[index].key > this->arr[parent].key){**

**swap(this->arr[index], this->arr[parent]);**

**shiftUp(parent);**

**}**

**return;**

**}**

**void shiftDown(int index){**

**int maxIndex = -1;**

**int lChildIndex = index \* 2+1;**

**int rChildIndex = (index \* 2) + 2;**

**if(lChildIndex < totalItems){**

**if(arr[index].key < arr[lChildIndex].key){**

**maxIndex = lChildIndex;**

**}**

**}**

**if(rChildIndex < totalItems){**

**int newindex = (maxIndex == -1 ? index : maxIndex);**

**if(arr[newindex].key < arr[rChildIndex].key){**

**maxIndex = rChildIndex;**

**}**

**}**

**if(maxIndex == -1){**

**return;**

**}**

**swap(arr[index], arr[maxIndex]);**

**shiftDown(maxIndex);**

**}**

**public:**

**MaxHeap(){**

**this->arr = NULL;**

**this->capacity = 0;**

**this->totalItems = 0;**

**}**

**MaxHeap(int \_capacity){**

**assert(\_capacity >= 1);**

**this->arr = new node[\_capacity];**

**this->capacity = \_capacity;**

**this->totalItems = 0;**

**}**

**void insert(int key,string msg,int date){**

**if(this->totalItems == this->capacity){**

**doubleCapacity();**

**}**

**this->arr[totalItems].key = key;**

**this->arr[totalItems].msg = msg;**

**this->arr[totalItems].date = date;**

**shiftUp(totalItems);**

**this->totalItems++;**

**//** **cout<<this->totalItems;**

**}**

**void getMax(){**

**if (totalItems != 0){**

**cout<<endl<<"Higher number has high priority"<<endl<<endl;**

**cout<<"Priority number is :"<<this->arr[0].key<<endl<<endl<<endl;**

**cout<<" Message is :"<<this->arr[0].msg<<endl<<endl<<endl;**

**cout<< "Date is """<<this->arr[0].date<<endl<<endl<<endl;}**

**}**

**void deleteMax()**

**{**

**assert(totalItems != 0);**

**swap(arr[0], arr[this->totalItems-1]);**

**totalItems--;**

**shiftDown(0);**

**}**

**bool isEmpty() const**

**{**

**return (totalItems == 0);**

**}**

**void deleteAll()**

**{**

**if (this->arr != NULL)**

**{**

**delete[]arr;**

**arr = NULL;**

**this->capacity = 0;**

**this->totalItems = 0;**

**}**

**}**

**~MaxHeap(){**

**deleteAll();**

**}**

**};**

**int main(){**

**MaxHeap heap(40);**

**heap.insert(24,"hello",29) ;**

**heap.insert(22,"hi",30);**

**heap.insert(40,"hlo",30);**

**heap.insert(37,"hlo",30);**

**heap.insert(39,"hlo",30);**

**heap.insert(38,"hlo",30);**

**int i ;**

**cout<<"Press 1 for getting online "<<endl;**

**cin >> i;**

**if (i==1){**

**do**  **{**

**if (heap.totalItems!=0){**

**heap.getMax();**

**heap.deleteMax();**

**}**

**cout<<endl<<"For closing the messenger Press 0 :";**

**cin>>i;**

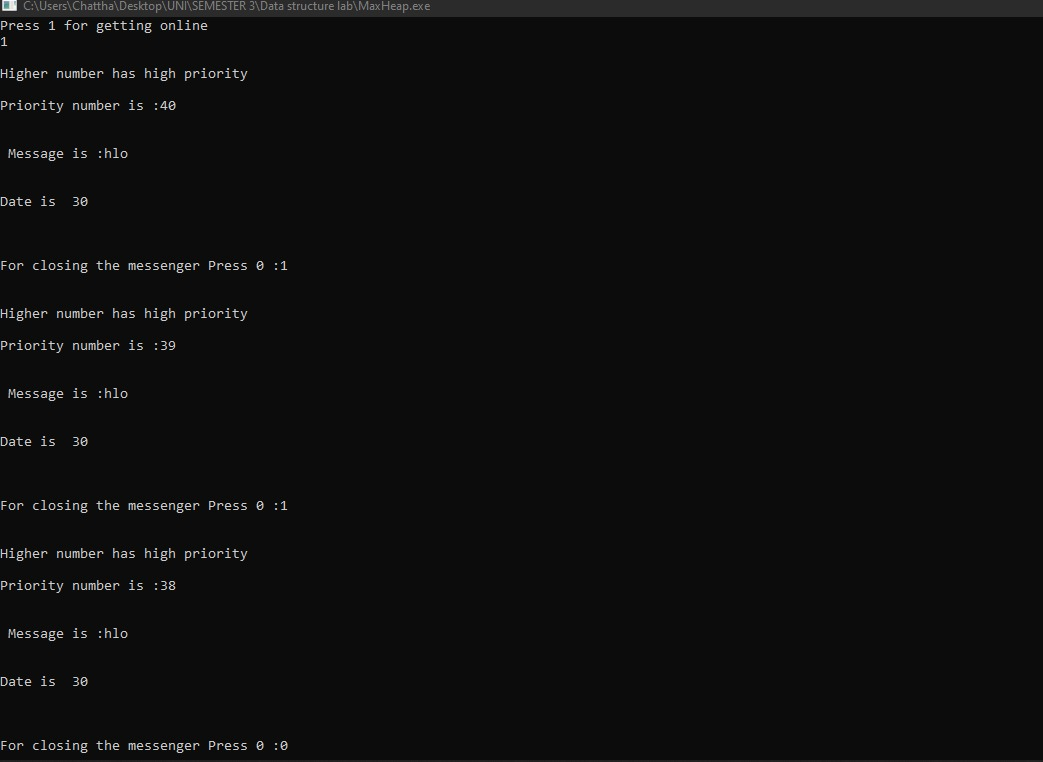
**cout<<endl;**

**}while(i==1);**

**}**

**}**

**Output:**



**Question 2:**

**Code:**

#include<iostream>

using namespace std;

class BST {

public:

int size;

int\* array;

void deletion ()

{

for (int i=size-1;i>=0;i--)

{

if(array[i]!='\0')

{

cout<<array[i]<<" is deleted"<<endl;

array[i]='\0';

return ;

}

}

}

int node\_at\_height(int root,int gheight,int bstheight,int height){

if(height==gheight)

{

if(array[root]!='\0'){

return 1;

}

else return 0;

}

if (height>bstheight)

{

return 0;

}

int leftchild=2\*root+1;

int rightchild=2\*root+2;

int left= node\_at\_height(leftchild,gheight,bstheight,height+1);

int right=node\_at\_height(rightchild,gheight,bstheight,height+1);

return left+right;

}

int Height(int root)

{

if (array[root]=='\0'){

return 0;

}

int leftchild=2\*root+1;

int rightchild=2\*root+2;

int leftheight=Height(leftchild);

int rightheight=Height(rightchild);

if(leftheight>rightheight){

return leftheight+1;

}

else{

return rightheight+1;

}

}

void leafnode()

{

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

{

int leftchild=2\*i+1;

int rightchild=2\*i+2;

if(leftchild<=size && rightchild<=size)

{

if(array[leftchild]=='\0' && array[rightchild]=='\0')

{

if(array[i]!='\0')

{

cout<<"Lead Node :"<<array[i]<<endl;

}

}

}

}

}

int extendSize(int x) {

int value = 0;

for(int y = 0; y < x + 1; y++) {

value = (2 \* value) + 2;

}

return value;

}

BST (int size) {

this -> size = extendSize(size);

this -> array = new int[this -> size];

for(int x = 0; x < this -> size; x++){

array[x] = '\0';

}

}

void insertElement(int x) {

int currentIndex = 0;

cout << "Adding: " << x;

while(true) {

if(array[currentIndex] == '\0'){

array[currentIndex] = x;

cout << " Inserted at index: " << currentIndex << endl;

break;

}else if(array[currentIndex] <= x) {

if(array[currentIndex] == x){

cout << "Repeating element" << endl;

break;

}else

cout << " Right ";

currentIndex = (2 \* currentIndex + 2);

}else if(array[currentIndex] >= x) {

if(array[currentIndex] == x){

cout << "Repeating element" << endl;

break;

}else

cout << " Left ";

currentIndex = (2 \* currentIndex + 1);

}

}

}

void searchElement(int x){

int currentIndex = 0;

while (true) {

if (array[currentIndex] == '\0') {

cout << "Not Found" << endl;

break;

}

if (array[currentIndex] == x) {

cout << "Found at index: " << currentIndex << endl;

break;

}

else if(array[currentIndex] < x) {

currentIndex = (2 \* currentIndex + 2);

}

else if(array[currentIndex] > x) {

currentIndex = (2 \* currentIndex + 1);

}

}

}

void parent(int x){

while (x != 0) {

x = (x-1) / 2;

}}

void inOrder(int currentIndex){

if(array[currentIndex] != '\0') {

inOrder(2 \* currentIndex + 1);

parent(currentIndex);

cout << array[currentIndex] << endl;

inOrder(2 \* currentIndex + 2);

}

}

void postOrder(int currentIndex) {

if(array[currentIndex] != '\0'){

postOrder(2 \* currentIndex + 1);

postOrder(2 \* currentIndex + 2);

parent(currentIndex);

cout << array[currentIndex] << " " << endl;

}

}

void preOrder(int currentIndex) {

if(array[currentIndex] != '\0') {

preOrder(2 \* currentIndex + 1);

parent(currentIndex);

cout << array[currentIndex] << " " << endl;

preOrder(2 \* currentIndex + 2);

}

}

};

int main () {

BST arr\_bst(5);

int bst\_height=0;

arr\_bst.insertElement(9);

arr\_bst.insertElement(2);

arr\_bst.insertElement(10);

arr\_bst.insertElement(11);

arr\_bst.insertElement(3);

arr\_bst.insertElement(1);

arr\_bst.inOrder(0);

arr\_bst.leafnode();

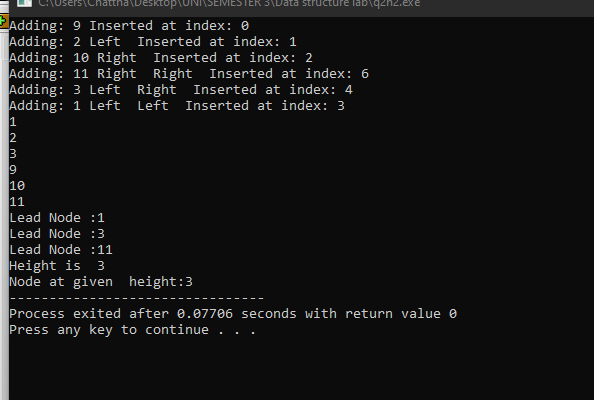
bst\_height=arr\_bst.Height(0);

cout<<"Height is "<< bst\_height<<endl;

cout<<"Node at given height:"<<arr\_bst.node\_at\_height(0,2,bst\_height,0);

};

**Output:**



**Question 3:**

**Code:**

#include<iostream>

using namespace std;

void Number\_of\_ways(int numner\_of\_stairs,int n, int a[]);

int main(){

int number\_of\_stairs;

cout<<"Enter number of stairs:";

cin>>number\_of\_stairs;

int a[number\_of\_stairs];

Number\_of\_ways(number\_of\_stairs, 0, a);

return 0;

}

void Number\_of\_ways(int number\_of\_stairs,int n, int a[]){

if(number\_of\_stairs == 0 || n > number\_of\_stairs) {

return;

}

if(n == number\_of\_stairs){

for(int i = 0 ; i < number\_of\_stairs; i++) {

if(a[i] != 0){

cout<<a[i]<<" ";

} }

cout<<endl; }

for (int i = 1; i < number\_of\_stairs + 1; i++){

a[n] = i;

Number\_of\_ways(number\_of\_stairs, n + i, a);

a[n] = 0;

}

}

**Output:**

