Homework 3

Question 1:

#include <assert.h>

#include<ctime>

#include<iostream>

using namespace std;

class MaxHeap

{

private:

        // Can be node class

    struct Node

    {

        int  key;

        int Date;

        string Msg;

        int Priority;

    };

    Node \*arr;

    int capacity; // Size of Heap

    int totalItems; //

    void doubleCapacity() // Increase the size

    {

        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];

        }

        if (this->arr!=NULL)

            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//date > this->arr[parent].date) && this.arr[index].pri > this.arr[parent].pri

        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:

bool CheckStatus(bool user)

{

    if(user==false)

    {

    cout<<"Offline : No Messages are displayed"<<endl;

    return false;

    }

    else

    {

    cout<<"Online : Messages are :"<<endl;

    return true;

    }

}

    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)

    {

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

        {

            doubleCapacity();

        }

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

        shiftUp(totalItems);

        this->totalItems++;

    }

    void getMax(int & key)

    {

        assert(totalItems != 0);

        key = this->arr[0].key;

    }

//  void priorityCheck(int Date,string Msg,int Priority)

//  {

//  string Messages[totalItems]={"Hello","Hy","AOA"};

//  for(int i =0;i<totalItems;i++)

// {

//  cout<<Messages[capacity]<<endl;

// }

// int PriorityArr[]={5,3,1};

// cout<<"Enter Message priority:"<<endl;

//  for(int i =0; i<totalItems;i++)

// {

//      // cin>>Priority;

//  cout<<PriorityArr[i]<<endl;

// }

void fun(int Date,string Msg,int Priority)

{

    //current date/time based on current system

   time\_t now = time(0);

   //convert now to string form

   char\* dt = ctime(&now);

   cout << "The local date and time is: " << dt << endl;

//    time\_t curr\_time;

//  curr\_time = time(NULL);

//  char \*tm = ctime(&curr\_time);

    int priority;

    string msg;

    string Messages[totalItems]={"Hello","Hy","AOA"};

int PriorityArr[]={5,3,1};

int i;

int index;

cout << "Read User Input into Array"<<endl;

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

{

    cout<<"Enter Priority:"<<endl;

    cin>>priority;

    cout<<"Enter Message:"<<endl;

    cin>>msg;

    while(PriorityArr[0]<PriorityArr[1])

    {

        index++;

        if(PriorityArr[0]<PriorityArr[1])

        swapPri(PriorityArr[0],PriorityArr[1],PriorityArr);

    }

    for(int i =0; i<totalItems;i++)

{

        // cin>>Priority;

    cout<<PriorityArr[i]<<endl;

}

while(Messages[0]<Messages[1])

    {

        index++;

        if(Messages[0]<Messages[1])

        swapMsg(0,1,Messages);

    }

    for(int i =0; i<totalItems;i++)

{

        // cin>>Priority;

    cout<<Messages[i]<<endl;

}

}

// CheckStatus(true)){

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

{

    received\_fun(Messages[i]);

}

    // }

}

int timestamp()

{

    int msg1,msg2;

//So compare timestamps.

    if (msg1.timestamp < msg2.timestamp)

        return -1;

    if (msg1.timestamp == msg2.timestamp)

        return 0;

    return 1;

}

void swapMsg(int msg1,int msg2,string Messages[])

{

    string temp = Messages[msg1];

    Messages[msg1]=Messages[msg2];

    Messages[msg2]=temp;

}

void swapPri(int a,int b,int PriorityArr[])

{

    int temp = PriorityArr[a];

    PriorityArr[a]=PriorityArr[b];

    PriorityArr[b]=temp;

}

void received\_fun(string messages){

    cout<<"THe message we got : "+ messages<<endl;

}

// void compare(string msg1,string  msg2)

// {

//  int priority;

// // returns -1, 0, 1 to indicate if msg1 is less than, equal to,

// // or greater than msg2.

//     if (msg1.priority == true)

//     {

//         if (msg2.priority == false)

//             return -1; // msg1 has priority flag set and msg2 doesn't

//     }

//     else if (msg2.priority == true)

//         return 1; // msg2 has priority flag set and msg1 doesn't

//     // At this point, we know that the priority flag is the same

//     // for both messages.

//     // So compare timestamps.

//     if (msg1.timestamp < msg2.timestamp)

//         return -1;

//     if (msg1.timestamp == msg2.timestamp)

//         return 0;

//     return 1;

// }

    };

int main()

{

    MaxHeap obj;

    obj.fun();

return 0;

}

Question 2:

#include<iostream>

using namespace std;

char Arr[100];

int arr\_size=100;

int root(char key) {

if (Arr[0] != '\0')

    cout << "Tree already had root";

else

    Arr[0] = key;

return 0;

}

int LeftTree(char key, int index) {

if (Arr[index] == '\0')

    cout << "set child at "

    << (index \* 2) + 1

    << " , no index found";

else

    Arr[(index \* 2) + 1] = key;

return 0;

}

int RightTree(char key, int index) {

if (Arr[index] == '\0')

    cout << "set child at "

    << (index \* 2) + 2

    << " , no index found";

else

    Arr[(index \* 2) + 2] = key;

return 0;

}

bool Search(int ele)

{

    int curr = 0;

    while(curr < arr\_size)

    {

         if(Arr[curr] == false)

          return false;

        if(Arr[curr] == ele)

            return true;

        else if(Arr[curr] < ele)

            curr = 2\*curr + 1;

         else

          curr = 2\*curr + 2;

    }

        return false;

  }

  void preorder(int index)

{

    if(index>0 && Arr[index]!='\0')

    {

        cout<<Arr[index]<<endl;

        preorder(LeftTree(index));

        preorder(RightTree(index));

    }

}

void postorder(int index)

{

    if(index>0 && Arr[index]!='\0')

    {

        postorder(LeftTree(index));

        postorder(RightTree(index));

        cout<<Arr[index]<<endl;

    }

}

void inorder(int index)

{

    if(index>0 && Arr[index]!='\0')

    {

        inorder(LeftTree(index));

        cout<<Arr[index]<<endl;

        inorder(RightTree(index));

    }

}

int FindHeight(int parent[], int n)

{

    int depth[n];

    for (int i = 0; i < n; i++)

        depth[i] = 0;

    for (int i = 0; i < n; i++)

        fillDepth(parent, i, depth);

}

void fillDepth(int parent[], int i, int depth[])

{

    if (depth[i])

        return;

    if (parent[i] == -1) {

        depth[i] = 1;

        return;

    }

    if (depth[parent[i]] == 0)

        fillDepth(parent, parent[i], depth);

    depth[i] = depth[parent[i]] + 1;

}

int LeafCount()

{

    int index;

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

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

    if( Arr[index] == '\0')

        return 0;

    if(Arr[lChildIndex] == NULL && Arr[rChildIndex] == NULL)

        return 1;

    else

        return LeafCount(Arr[lChildIndex])+

            LeafCount(Arr[rChildIndex]);

}

int Display() {

cout << "\n";

for (int i = 0; i < 100; i++) {

    if (Arr[i] != '\0')

    cout << Arr[i];

    else

    cout << "-";

}

return 0;

}

int main() {

root(16);

LeftTree(3,0);

RightTree(8, 0);

LeftTree(9, 1);

RightTree(2, 1);

RightTree(7, 2);

Display();

return 0;

}

Question 3:

/\*

First of all recursion to get how many combination will be genrated and then we use recursion function --arr() which will give us arr which has all the value and combination

and we passed in binary search tree class array Then we use heap the concept array to make a binary tree

and then used level by  level traversal to print the series of cobbination.

\*/

#include <iostream>

using namespace std;

int countWays(int n)

{

    int res[n + 1];

    res[0] = 1;

    res[1] = 1;

    res[2] = 2;

    for (int i = 3; i <= n; i++)

        res[i] = res[i - 1] + res[i - 2] + res[i - 3];

    return res[n];

}

// This code is contributed by shubhamsingh10

class node{

    public:

    int data;

    node\* left, \*right ,\*next;

    int i=0;

};

class BinarySearchTree{

        public:

    node obj3;

    node\* root;

        int arr[20];

    node\* Insert( node\* root, int val);

    void    Insert(int val){

       Insert(this->root,  val);}

    int fun(int n,int i,int p )

    {

        if(i!=(p+1))

    {

        arr[i]=n;

    }

    if(n==1)

    {

        return 1;

    }else if(n<0)

    {

        return 0;

    }else

    {

        return fun(n-1,i=i+1,p),fun(n-(n-1),i=i+1,p),fun(n-(n-1),i=i+1,p);

    }

    }

    node\* BinarySearchTree::Insert(node\* r, int val )

    {

 if (r==NULL)

    {

        node\* t= new node;

        if (r==root)

            root=t;

        else

        r=t;

    }

    else if (r->data== val){

        cout<<"Record already exist"<<val;

    }

    else if (val < r->data) // insert on left s

    r->left = Insert(r->left , val );

    else if (val > r->data)

        r->right= Insert( r->right,val);

    return r;

    }

    void print(node\*temp,int level,int n)

    {

    if(n=level)

    {

        cout<<temp->next;

        return;

    }

    if(temp->left != NULL)

    {

        print(temp->left,level,n+1);

    }

    if(temp->right != NULL)

    {

        print(temp->right,level,n+1);

    }

    }

    void levelTraversal(node\*temp,int level)

    {

    for(int i=0;i<level;i++)

    {

        print(temp,i,0);

        cout<<endl;

    }

    }

    void array\_01()

    {

    for(int j=0; j>10; j++ ){

        cout<<arr[j];

    }

    }

};

int main()

{

    // Write C++ code here

    BinarySearchTree obj5;

    node obj9;

    int s=0;

    s=countWays(4);

   obj5 .fun(4,0,s);

    obj5.array\_01();

    for(int i=0; i<s; i++ )

    {

      obj5.Insert(obj5.root,obj5.arr[i]);

    }

    return 0;

}

The end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_