Stacks

Stacks Class -1

```cpp
#include <iostream>
#include<stack>

using namespace std;

int main()
{
    stack<int>st;
    st.push(10);
    st.push(20);
    st.push(30);
    st.push(40);

    st.pop();
    cout<<"top element :"<<st.top()<<endl;

    cout<<"stack size :"<<st.size()<<endl;

    if(st.empty()){
        cout<<"stack is empty "<<endl;
    }
    else{
        cout<<"stack is not empty "<<endl;
    }
    return 0;
}
```

```
PS E:\C++Code> g++ .\f7.cpp
PS E:\C++Code> .\a.exe
top element :30
stack size :3
stack is not empty
```

```cpp
#include <iostream>
#include<stack>

using namespace std;

int main()
{
    stack<int>st;
    st.push(10);
    st.push(20);
    st.push(30);
    st.push(40);

    while(!st.empty()){
        cout<<st.top()<<" ";
        st.pop();
    }
    cout<<endl;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f7.cpp
PS E:\C++Code> .\a.exe
40 30 20 10
```

```cpp
#include <iostream>
#include<stack>

using namespace std;

class Stack{
    private:
    int *arr;
    int top;
    int size;

    public:
    Stack(int size){
        arr=new int[size];
        this->size=size;
        top=-1;
    }

    void push(int data ){
        if(size-top>1){
            top++;
            arr[top]=data;
        }
        else{
            cout<<"Stack Overflow"<<endl;
        }
    }

    void pop(){
        if(top==-1){
            cout<<"stack underflow, cant delete element"<<endl;
        }
        else{
            top--;
        }
    }

    int getTop(){
        if(top==-1){
            cout<<"there is no element"<<endl;
        }
        else{
            return arr[top];
        }
    }
```

```
    int getSize(){
        return top+1;
    }

    bool isEmpty(){
        if(top==-1){
            return true;
        }
        else{
            return false;
        }
    }
};

int main()
{
    Stack s(10);
    s.push(10);
    s.push(20);
    s.push(30);
    s.push(40);

    while(!s.isEmpty()){
        cout<<s.getTop()<<" ";
        s.pop();
    }
    cout<<endl;
    cout<<"size of stack "<< s.getSize()<<endl;

    return 0;
}
```

```
PS E:\C++Code> g++ .\f7.cpp
PS E:\C++Code> .\a.exe
40 30 20 10
size of stack 0
```

Q)Create Two stack in one array

```
#include <iostream>

using namespace std;

class Stack{
    public:
    int *arr;
    int size;
    int top1;
    int top2;

    Stack(int size){
        arr=new int[size];
        this->size=size;
        top1=-1;
        top2=size;
    }

    void push1(int data){
        if(top2-top1==1){
            cout<<"overflow stack 1"<<endl;
        }
        else{
            top1++;
            arr[top1]=data;
        }
    }
    void pop1(){
        if(top1==-1){
            cout<<"underFlow in stack 1"<<endl;
        }
        else{
             arr[top1]=0;

            top1--;
        }
    }
```

```
  void push2(int data){
        if(top2-top1==1){
          cout<<"overflow stack 2"<<endl;
       }
       else{
           top2--;
           arr[top2]=data;
       }
   }
   void pop2(){
       if(top2==size){
           cout<<"underFlow in stack 2"<<endl;
       }
       else{
            arr[top2]=0;

           top2++;
       }
   }

   void print(){
   cout<<endl;
   cout<<"top1: "<<top1<<endl;
   cout<<"top2: "<<top2<<endl;
   for(int i=0;i<size;i++){
       cout<<arr[i]<<" ";
   }
   cout<<endl;
}
};
```

```
int main()
{
    Stack st(10);
    st.push1(10);
    st.print();
    st.push1(20);
    st.print();
    st.push1(30);
    st.print();
    st.push1(40);
    st.print();
    st.push1(50);
    st.print();
    st.push1(60);
    st.print();

    st.push2(100);
    st.print();
    st.push2(110);
    st.print();
    st.push2(120);
    st.print();

    st.pop1();
    st.print();
    st.pop2();
    st.print();

  return 0;
}
```

```
PS E:\C++Code> .\a.exe

top1: 0
top2: 10
10 15668792 0 0 0 0 0 0 0 0

top1: 1
top2: 10
10 20 0 0 0 0 0 0 0 0

top1: 2
top2: 10
10 20 30 0 0 0 0 0 0 0

top1: 3
top2: 10
10 20 30 40 0 0 0 0 0 0

top1: 4
top2: 10
10 20 30 40 50 0 0 0 0 0

top1: 5
top2: 10
10 20 30 40 50 60 0 0 0 0

top1: 5
top2: 9
10 20 30 40 50 60 0 0 0 100

top1: 5
top2: 8
10 20 30 40 50 60 0 0 110 100

top1: 5
top2: 7
10 20 30 40 50 60 0 120 110 100
```

Pop

```
top1: 4
top2: 7
10 20 30 40 50 0 0 120 110 100

top1: 4
top2: 8
10 20 30 40 50 0 0 0 110 100
```

Q)reverse program using stack

```cpp
#include <iostream>
#include <stack>

using namespace std;

int main()
{
    string name="AnandKumar";
    stack<char>st;
    for(int i=0;i<name.length();i++){
        st.push(name[i]);
    }

    while(!st.empty()){
        cout<<st.top();
        st.pop();
    }
  return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
ramuKdnanA
```

Q) find middle element in stack

```cpp
#include <iostream>
#include <stack>

using namespace std;

void printMiddle(stack<int> &st, int &totalSize){
    if(st.size()==0){
        cout<<"there is no element "<<endl;
        return;
    }
    cout<<"st size()"<<st.size()<<endl;
    if(st.size()==totalSize/2+1){
        cout<<"middle element is "<<st.top()<<endl;
        return;
    }
    int temp=st.top();
    st.pop();

    printMiddle(st,totalSize);
    st.push(temp);
}

int main()
{
   stack<int>st;
   st.push(10);
   st.push(20);
   st.push(30);
   st.push(40);
   st.push(50);
   st.push(60);
   st.push(70);
   int totalSize=st.size();
   printMiddle(st,totalSize);
  return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
st size()7
st size()6
st size()5
st size()4
middle element is 40
```

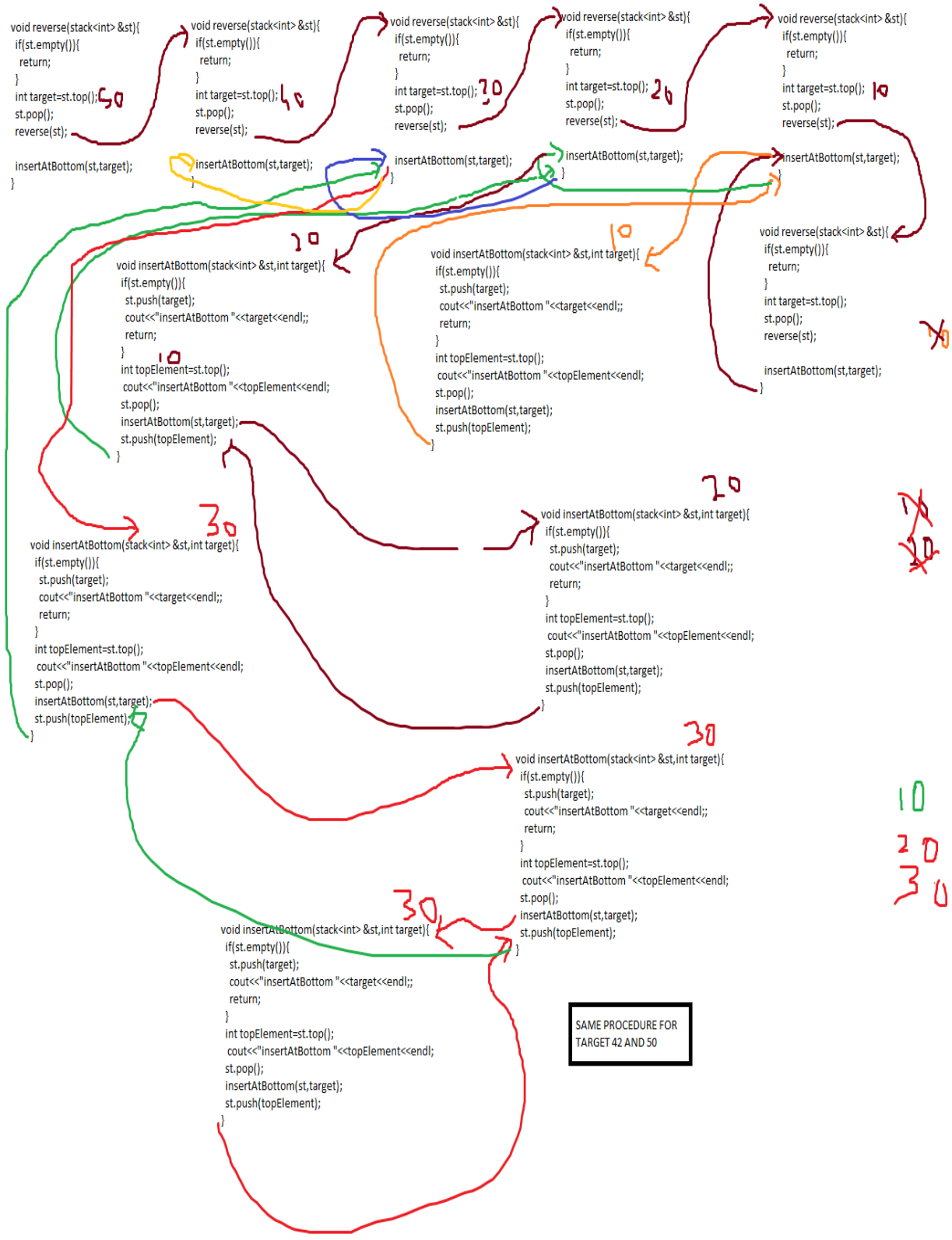
Stacks Class -2

Q) top insert at bottom

```cpp
#include <iostream>
#include <stack>
#include <algorithm>

using namespace std;

void solve (stack<int> &st,int target){

    if(st.empty()){
        st.push(target);
        return ;
    }

    int topElement=st.top();
    st.pop();
    solve(st,target);
    st.push(topElement);
}

void insertAtBottom(stack<int> &st){

    if(st.empty()){
        cout<<"stack is empty ,can't insert at bottom "<<endl;
        return ;
    }

    int target =st.top();
    st.pop();
    solve (st,target);
}
```

```cpp
int main()
{
    stack<int> st;
    st.push(10);
    st.push(20);
    st.push(30);
    st.push(40);
    st.push(50);
    insertAtBottom(st);

    while(!st.empty()){
        cout<<st.top()<<" ";
        st.pop();
    }
    cout<<endl;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
40 30 20 10 50
```


void reverse(stack<int> &st){
if(st.empty()){
return;
}
int target=st.top();
st.pop();
reverse(st);
insertAtBottom(st,target);
}
void insertAtBottom(stack<int> &st,int target){
if(st.empty()){
st.push(target);
cout<<"insertAtBottom "<<target<<endl;;
return;
}
int topElement=st.top();
cout<<"insertAtBottom "<<topElement<<endl;
st.pop();
insertAtBottom(st,target);
st.push(topElement);
}
SAME PROCEDURE FOR
TARGET 42 AND 50

20. Valid Parentheses(leetcode)

```cpp
#include <iostream>
#include <stack>

using namespace std;

bool isValid(string s)
{
  stack<char> st;
  for (int i = 0; i < s.length(); i++)
  {
    char ch = s[i];
    if (ch == '(' || ch == '{' || ch == '[')
    {
      st.push(ch);
    }
    else
    {
      if (!st.empty())
      {
        char topCh = st.top();
        if (ch == ')' && topCh == '(')
        {
          st.pop();
        }
        else if (ch == '}' && topCh == '{')
        {
          st.pop();
        }
        else if (ch == ']' && topCh == '[')
        {
          st.pop();
        }
        else
        {
          return false;
        }
      }
      else
      {
        return false;
      }
    }
  }
```

```
  if (st.empty())
  {
    return true;
  }
  else
  {
    return false;
  }
}
int main()
{
  string s = "{[]}";
  if (isValid(s))
  {
    cout << "it Is  valid";
  }
  else
  {
    cout << "it is Not valid";
  }
  return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
it Is  valid
```

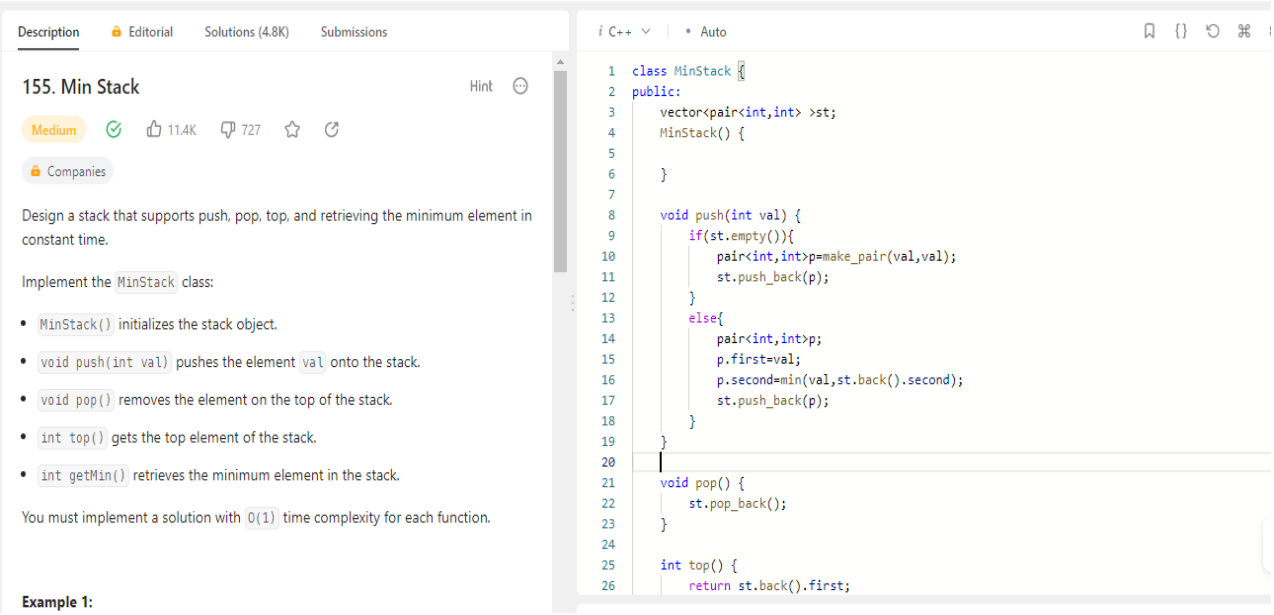
Q) sort stack(important)

```cpp
#include <iostream>
#include <stack>

using namespace std;

void insertedSort(stack<int> &st,int target){

    if(st.empty()){
        st.push(target);
        return;
    }

    if(st.top()>=target){
        st.push(target);
        return;
    }

    int topElement=st.top();
    st.pop();
    insertedSort(st,target);
    st.push(topElement);

}

void sortStack(stack<int> &st){

    if(st.empty()){
        return;
    }

    int topElement=st.top();
    st.pop();
    sortStack(st);

    insertedSort(st,topElement);
}
```

```cpp
int main()
{
  stack<int> st;
  st.push(7);
  st.push(11);
  st.push(3);
  st.push(5);
  st.push(9);

  sortStack(st);
  cout<<"print :"<<endl;
  while (!st.empty()){
    cout<<st.top()<<" ";
    st.pop();
  }
  return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
print :
3 5 7 9 11
```

Stack Class – 3 and 4 [HD Processing]



Description | Editorial | Solutions (4.8K) | Submissions

155. Min Stack

Hint

Medium 11.4K 727

Companies

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with `O(1)` time complexity for each function.

Example 1:

C++ | Auto

```
class MinStack {
public:
    vector<pair<int,int> >st;
    MinStack() {

    }

    void push(int val) {
        if(st.empty()){
            pair<int,int>p=make_pair(val,val);
            st.push_back(p);
        }
        else{
            pair<int,int>p;
            p.first=val;
            p.second=min(val,st.back().second);
            st.push_back(p);
        }
    }

    void pop() {
        st.pop_back();
    }

    int top() {
        return st.back().first;
```

```
class MinStack {
public:
    vector<pair<int,int> >st;
    MinStack() {

    }

    void push(int val) {
        if(st.empty()){
            pair<int,int>p=make_pair(val,val);
            st.push_back(p);
        }
        else{
            pair<int,int>p;
            p.first=val;
            p.second=min(val,st.back().second);
            st.push_back(p);
        }
    }

    void pop() {
        st.pop_back();
    }
```

```
    int top() {
        return st.back().first;
    }

    int getMin() {
        return st.back().second;
    }
};
```

---

32. Longest Valid Parentheses

Hard 10.9K 347

Companies

Given a string containing just the characters `'('` and `')'`, return *the length of the longest valid (well-formed) parentheses substring*.

Example 1:

```
Input: s = "(()"
Output: 2
Explanation: The longest valid parentheses substring is "()".
```

Example 2:

```
Input: s = ")()())"
Output: 4
Explanation: The longest valid parentheses substring is "()()".
```

```
1  class Solution {
2  public:
3      int longestValidParentheses(string s) {
4          stack<int>st;
5          st.push(-1);
6
7          int maxlen=0;
8          for(int i=0;i<s.length();i++){
9             char ch=s[i];
10            if(ch=='('){
11                st.push(i);
12            }
13            else{
14                st.pop();
15                if(st.empty()){
16                    st.push(i);
17                }
18                else{
19                    int len=i-st.top();
20                    maxlen=max(len,maxlen);
21                }
22            }
23         }
24         return maxlen;
25     }
26 };
```

Console

Q)Next smaller element

```cpp
#include <iostream>
#include <stack>
#include <vector>

using namespace std;

int main()
{
    vector<int>v;
    v.push_back(2);
    v.push_back(1);
    v.push_back(4);
    v.push_back(3);

    stack<int>st;
    st.push(-1);

    vector<int>ans(v.size());

    for(int i=v.size()-1;i>=0;i--){
        int curr=v[i];
        while (st.top()>=curr)
        {
            st.pop();
        }
        ans[i]=st.top();
        st.push(curr);
    }
    cout<<"printing "<<endl;
    for(int i=0;i<ans.size();i++){
        cout<<ans[i]<<" ";
    }
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
printing
1 -1 3 -1
```

Q) previous smaller elements

```cpp
#include <iostream>
#include <stack>
#include <vector>

using namespace std;

vector<int> previousSmallerElement(vector<int> &v){

    stack<int>st;
    st.push(-1);

    vector<int>ans(v.size());

    for(int i=0;i<v.size();i++){
        int curr=v[i];
        while (st.top()>=curr)
        {
            st.pop();
        }
        ans[i]=st.top();
        st.push(curr);
    }
    return ans;
}

int main()
{
    vector<int>v;
    v.push_back(2);
    v.push_back(1);
    v.push_back(4);
    v.push_back(3);

    vector<int>ans=previousSmallerElement(v);

    cout<<"printing previous smaller elements :"<<endl;
    for(int i=0;i<ans.size();i++){
        cout<<ans[i]<<" ";
    }
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
printing previous smaller elements :
-1 -1 1 1
```

Time complexity :O(n)

Q) get largest number in Histogram(leetCode 84)

```cpp
#include <iostream>
#include <vector>
#include <stack>
#include <limits.h>

using namespace std;

vector<int> preSmallerElement(vector<int> &input){
    stack<int> s;
    s.push(-1);

    vector<int> ans(input.size());

    for(int i=0;i<input.size();i++){
        int curr=input[i];

        while (s.top()!=-1 && input[s.top()]>=curr)
        {
            s.pop();
        }
        ans[i]=s.top();
        s.push(i);
    }
    return ans;
}
vector<int> nextSmallerElement(vector<int> &input){
    stack<int> s;
    s.push(-1);

    vector<int> ans(input.size());

    for(int i=input.size()-1;i>=0;i--){
        int curr=input[i];

        while (s.top()!=-1 && input[s.top()]>=curr)
        {
            s.pop();
        }
        ans[i]=s.top();
        s.push(i);
    }
    return ans;
}
```

```cpp
int largestRectangularArea(vector<int> &heights){
    vector<int> prev=preSmallerElement(heights);
    vector<int> next=nextSmallerElement(heights);

    int maxArea=INT_MIN;
    int size=heights.size();

    for(int i=0;i<heights.size();i++){
        int length=heights[i];

        if(next[i]==-1){
            next[i]=6;
        }
        int width=next[i]-prev[i]-1;
        int area=length* width;
        maxArea=max(maxArea,area);
    }
    return maxArea;
}
int main()
{
   vector<int>v;
   v.push_back(2);
   v.push_back(1);
   v.push_back(5);
   v.push_back(6);
   v.push_back(2);
   v.push_back(3);

   cout<<"ans is: " <<largestRectangularArea(v);
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
ans is: 10
```

Queues

Queue – Class 1

```cpp
#include <iostream>
#include <queue>
using namespace std;


int main() {
   queue<int>q;
   q.push(10);
   q.push(20);
   q.push(30);
   q.push(40);
   q.push(50);

   cout<<"queue size :"<<q.size()<<endl;
   q.pop();

   cout<<"after pop queue size :"<<q.size()<<endl;

   if(q.empty()){
    cout<<"queue is empty "<<endl;
   }
   else{
    cout<<"queue is NOT empty "<<endl;
   }

   cout<<"top element is :"<<q.front()<<endl;

}
```

```
PS E:\C++Code> .\a.exe
queue size :5
after pop queue size :4
queue is NOT empty
top element is :20
```

Q)create push ,pop ,getfront, isEmpty, getFront Method in Queue

```cpp
#include <iostream>
#include <queue>
using namespace std;

class Queue{

    public:
    int* arr;
    int size;
    int front;
    int rear;

    Queue(int size){
        this->size=size;
        arr=new int[size];
        front=0;
        rear=0;
    }

    void push(int data){
        if(rear==size){
            cout<<"Q is full "<<endl;
        }
        else{
            arr[rear]=data;
            rear++;
        }
    }

    void pop(){
        if(front==rear){
            cout<<"Q is empty "<<endl;
        }
        else{
            arr[front]=-1;
            front++;
            if(front==rear){
                front=0;
                rear=0;
            }
        }
    }
```

```
    int getFront(){
        if(front==rear){
            cout<<"Q is empty "<<endl;
        }
        else{
            return arr[front];
        }
    }

    bool isEmpty(){
         if(front==rear){
            return true;
        }
        else{
            return false;
        }
    }

    int getSize(){
        return rear-front;
    }
};

int main()
{
   Queue q(2);
   q.push(5);
   q.push(7);
   q.push(9);
   cout<<"size is "<<q.getSize()<<endl;
   q.pop();
   cout<<"after pop size is "<<q.getSize()<<endl;
   cout<<"get front element "<<q.getFront()<<endl;

   if(q.isEmpty()){
    cout<<"Q is empty";
   }
   else{
    cout<<"Q is Not empty";
   }
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
Q is full
size is 2
after pop size is 1
get front element 7
Q is Not empty
```

Q) circular queue

```
#include <iostream>
#include <queue>

using namespace std;

class CirQueue{

    public:
    int size;
    int *arr;
    int front;
    int rear;

    CirQueue(int size){
        this->size=size;
        arr=new int[size];
        front=-1;
        rear=-1;
    }

    void push(int data){
        if(front==0 && rear==size-1){
            cout<<"Q is full, cannot insert "<<endl;
        }
        else if(front==-1){
            front=rear=0;
            arr[rear]=data;
        }
        else if(rear==size-1 && front!=0){
            rear=0;
            arr[rear]=data;
        }
        else{
            rear++;
            arr[rear]=data;
        }
    }
```

```
    void pop(){
        if(front==-1){
            cout<<"Q is empty ,cannot pop "<<endl;
        }
        else if(front==rear){
            arr[front]=-1;
            front=-1;
            rear=-1;
        }
        else if(front ==size-1){
            front=0;
        }
        else{
            front++;
        }

    }

};

int main()
{

    return 0;
}
```

Q) Doubly Circular deque

```
#include <iostream>
#include <queue>

using namespace std;

class Deque
{
public:
    int size;
    int *arr;
    int front;
    int rear;

    Deque(int size)
    {
        this->size = size;
        arr = new int[size];
        front = -1;
        rear = -1;
    }

    void pushRear(int data)
    {
        if (front == 0 && rear == size - 1)
        {
            cout << "Q is full, cannot insert " << endl;
            return;
        }
        else if (front == -1)
        {
            front = rear = 0;
        }
        else if (rear == size - 1 && front != 0)
        {
            rear = 0;
        }
        else
        {
            rear++;
        }
        arr[rear] = data;
    }
```

```
void pushFront(int data)
{
    if (front == 0 && rear == size - 1)
    {
        cout << "Q is full, cannot insert " << endl;
        return;
    }
    else if (front == -1)
    {
        front = rear = 0;
    }
    else if (front == 0 && rear != size - 1)
    {
        front = size - 1;
    }
    else
    {
        front--;
    }
    arr[front] = data;
}

void popFront()
{
    if (front == -1)
    {
        cout << "Q is empty ,cannot pop " << endl;
    }
    else if (front == rear)
    {
        arr[front] = -1;
        front = -1;
        rear = -1;
    }
    else if (front == size - 1)
    {
        arr[front] = -1;
        front = 0;
    }
    else
    {
        arr[front] = -1;
        front++;
    }
}
```

```
    void popRear()
    {
        if (front == -1)
        {
            cout << "Q is empty ,cannot pop " << endl;
        }
        else if (front == rear)
        {
            arr[front] = -1;
            front = -1;
            rear = -1;
        }
        else if (rear == 0)
        {
            arr[rear] = -1;
            rear = size - 1;
        }
        else
        {
            arr[rear] = -1;
            rear--;
        }
    }

    void print()
    {
        // print
        for (int i = 0; i < size; i++)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};
int main()
{
    Deque dq(10);
    dq.print();
    dq.pushRear(10);
    dq.pushRear(20);
    dq.pushRear(30);
    dq.pushRear(40);
    dq.pushRear(50);
    dq.print();
```

```
    dq.popRear();
    dq.popRear();
    dq.print();

    dq.pushFront(101);
    dq.pushFront(102);
    dq.pushFront(103);
    dq.pushFront(104);
    dq.pushFront(105);
    dq.pushFront(106);
    dq.print();

    dq.popFront();
    dq.print();
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
18706432 18683448 0 0 0 0 0 0 0 0
10 20 30 40 50 0 0 0 0 0
10 20 30 -1 -1 0 0 0 0 0
10 20 30 -1 106 105 104 103 102 101
10 20 30 -1 -1 105 104 103 102 101
```

Queue – Class 2

Q) reverse using queue

```cpp
#include <iostream>
#include <stack>
#include <queue>

using namespace std;

void reverseQueue(queue<int> &q){

    stack<int> st;

    while (!q.empty())
    {
        int element =q.front();
        q.pop();

        st.push(element);
    }

    while(!st.empty()){

        int element=st.top();
        st.pop();

        q.push(element);
    }

}
int main()
{
    queue<int> q;
    q.push(1);
    q.push(2);
    q.push(3);
    q.push(4);
    q.push(5);

    reverseQueue(q);
```

```
    while (!q.empty())
    {
        cout<<q.front()<<" ";
        q.pop();
    }
    cout<<endl;

    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
5 4 3 2 1
```

Tc =O(n)

Sc=O(n)

Q)reverse queue using recursion

```
#include <iostream>
#include <stack>
#include <queue>

using namespace std;

void reverseQueueRecurrsion(queue<int> &q){

   if(q.empty()){
        return;
   }

   int element=q.front();
   q.pop();

   reverseQueueRecurrsion(q);
   q.push(element);
}

int main()
{
    queue<int> q;
    q.push(1);
    q.push(2);
    q.push(3);
    q.push(4);
    q.push(5);

    reverseQueueRecurrsion(q);

    while (!q.empty())
    {
        cout<<q.front()<<" ";
        q.pop();
    }
    cout<<endl;

    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
5 4 3 2 1
```

Tc =O(n)

Sc=O(n)

Q) k reverse using queue

```cpp
#include <iostream>
#include <stack>
#include<queue>

using namespace std;

void reverseKqueue(queue<int> &q,int k){

    stack<int>st;
    int count=0;
    int n=q.size();

    if(k<=0 && k>n){
        return;
    }

    while (!q.empty())
    {
        int temp=q.front();
        q.pop();

        st.push(temp);
        count++;
        if(count==k){
            break;
        }
    }

    while(!st.empty()){
        int temp=st.top();
        st.pop();
        q.push(temp);
    }
```

```
    count =0;
    while(!q.empty() && n-k!=0){
        int temp=q.front();
        q.pop();
        q.push(temp);
        count++;

        if(n-k==count){
            break;
        }
    }
}

int main()
{
    queue<int>q;
    q.push(10);
    q.push(20);
    q.push(30);
    q.push(40);
    q.push(50);

    int k=3;
    reverseKqueue(q,k);

    while (!q.empty())
    {
       cout<<q.front()<<" ";
       q.pop();
    }
    cout<<endl;

    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
30 20 10 40 50
```

Q) interleave queue

```
#include <iostream>
#include <stack>
#include<queue>

using namespace std;

void interLeaveQueue(queue<int> &q){

    int n=q.size();

    if(q.empty()){
        return ;
    }

    int k=n/2;
    int count =0;

    queue<int> q2;

    while(!q.empty()){
        int temp=q.front();
        q.pop();

        q2.push(temp);

        count++;
        if(count==k){
            break;
        }
    }

    while (!q.empty() && !q2.empty())
    {
        int first=q2.front();
        q2.pop();

        q.push(first);

        int second=q.front();
        q.pop();

        q.push(second);
    }
```

```
    if(n&1){
        int element=q.front();
        q.pop();

        q.push(element);
    }

}

int main()
{
    queue<int>q;
    q.push(1);
    q.push(2);
    q.push(3);
    q.push(4);
    q.push(5);
    q.push(6);
    q.push(7);
    q.push(8);
    q.push(9);

    interLeaveQueue(q);

    while (!q.empty())
    {
        cout<<q.front()<<" ";
        q.pop();
    }
    cout<<endl;

    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
1 5 2 6 3 7 4 8 9
```

Tc=O(1)

Sc=O(n)

Q) find –ve integer in every window of size k (Important)

```cpp
#include <iostream>
#include <deque>
#include<queue>

using namespace std;

void solve(int arr[],int size,int k){

    deque<int>q;

    for(int i=0;i<k;i++){
        if(arr[i]<0){
            q.push_back(i);
        }
    }

    for(int i=k;i<size;i++){
        if(q.empty()){
            cout<< 0 <<" ";
        }
        else{
            cout<<arr[q.front()]<<" ";
        }

        while((!q.empty()) && (i-q.front()>=k)){
            q.pop_front();
        }
        if(arr[i]<0){
            q.push_back(i);
        }
    }

    if(q.empty()){
        cout<< 0 <<" ";
    }
    else{
        cout<<arr[q.front()]<<" ";
    }
}
```

```cpp
int main()
{
    int arr[]={12,-1,-7,8,-15,30,16,28};
    int size=8;
    int k=3;

    solve(arr,size,k);
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
-1 -1 -7 -15 -15 0
```

Tc=O(n)

Sc=O(k)

Queue – Class 3

Q) first non-repeating character in stream(Imp)

```cpp
#include <iostream>
#include <queue>

using namespace std;

string solve(string str)
{
    int freq[26] = {0};
    queue<int> q;

    string ans = "";
    for (int i = 0; i < str.length(); i++)
    {
        char ch = str[i];
        freq[ch - 'a']++;
        q.push(ch);

        while (!q.empty())
        {
            if (freq[q.front() - 'a'] > 1)
            {
                q.pop();
            }
            else
            {
                ans.push_back(q.front());
                break;
            }
        }
        if (q.empty())
        {
            ans.push_back('#');
        }
    }

    return ans;
}
```

```cpp
int main()
{
    string str = "aabc";
    string ans = solve(str);
    cout << "no repeating elements :" << ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
no repeating elements :a#bb
```

Tc=O(n)

Sc=O(n)

---

134. Gas Station (Imp)

Medium 10K 844

Companies

T(= O(n)

There are `n` gas stations along a circular route, where the amount of gas at the `i^th` station is `gas[i]`.

You have a car with an unlimited gas tank and it costs `cost[i]` of gas to travel from the `i^th` station to its next `(i + 1)^th` station. You begin the journey with an empty tank at one of the gas stations.

Given two integer arrays `gas` and `cost`, return *the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return* `-1`. If there exists a solution, it is **guaranteed** to be **unique**

SC= O(1)

Example 1:

```
Input: gas = [1,2,3,4,5], cost = [3,4,5,1,2]
Output: 3
```

```cpp
public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
        int deficit=0;
        int balance =0;
        int start=0;

        for(int i=0;i<gas.size();i++){
            balance+=gas[i]-cost[i];
            if(balance<0){
                deficit+=balance;
                start=i+1;
                balance=0;
            }
        }

        if( deficit+balance>=0){
            return start;
        }
        else{
            return -1;
        }
    }
};
```

239. Sliding Window Maximum (Imp)

Hint

Hard 14.5K 470

Companies

You are given an array of integers `nums`, there is a sliding window of size `k` which is moving from the very left of the array to the very right. You can only see the `k` numbers in the window. Each time the sliding window moves right by one position.

Return *the max sliding window.*

Example 1:

```
Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [3,3,5,5,6,7]
Explanation:
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
```

```cpp
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        deque<int>dq;
        vector<int>ans;

        for(int i=0;i<k;i++){
            while(!dq.empty() && nums[i]>=nums[dq.back()]){
                dq.pop_back();
            }
            dq.push_back(i);
        }

        ans.push_back(nums[dq.front()]);

        for(int i=k;i<nums.size();i++){
            if(!dq.empty() && i-dq.front()>=k){
                dq.pop_front();
            }

            while(!dq.empty() && nums[i]>=nums[dq.back()]){
                dq.pop_back();
            }

            dq.push_back(i);
```

Console

Trees – I

Trees Class – 1

Q) make tree : 90 ,20 ,11 ,-1 ,-1 ,13 ,-1 ,-1 ,50 ,-1 ,-1

```cpp
#include <iostream>
#include <queue>

using namespace std;

class Node{

    public :
    int data ;
    Node* left;
    Node* right;

    Node(int data){
        this->data=data;
        left=NULL;
        right=NULL;
    }
};

Node* buildTree(){

    int data;
    cout<<"enter the data :"<<endl;
    cin>>data;

    if(data==-1){
        return NULL;
    }

    Node* root=new Node(data);

    cout<<"enter the data for left part "<<data<<" node"<<endl;
    root->left=buildTree();

    cout<<"enter the data for right part "<<data<<" node"<<endl;
    root->right=buildTree();

    return root;
}
```

```cpp
int main()
{
    Node* root=NULL;
    root=buildTree();
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data :
90
enter the data for left part 90 node
enter the data :
20
enter the data for left part 20 node
enter the data :
11
enter the data for left part 11 node
enter the data :
-1
enter the data for right part 11 node
enter the data :
-1
enter the data for right part 20 node
enter the data :
13
enter the data for left part 13 node
enter the data :
-1
enter the data for right part 13 node
enter the data :
-1
enter the data for right part 90 node
enter the data :
50
enter the data for left part 50 node
enter the data :
-1
enter the data for right part 50 node
enter the data :
-1
```

Q) tree using level ordered traversal

```
#include <iostream>
#include <queue>

using namespace std;

class Node{
    public :
    int data ;
    Node* left;
    Node* right;

    Node(int data){
        this->data=data;
        left=NULL;
        right=NULL;
    }
};

Node* buildTree(){
    int data;
    cout<<"enter the data :"<<endl;
    cin>>data;
    if(data==-1){
        return NULL;
    }

    Node* root=new Node(data);

    cout<<"enter the data for left part "<<data<<" node"<<endl;
    root->left=buildTree();

    cout<<"enter the data for right part "<<data<<" node"<<endl;
    root->right=buildTree();

    return root;
}
```

```cpp
void levelOrderedTraversal(Node* root){

    queue<Node*> q;

    q.push(root);
    while(!q.empty()){
        Node * temp=q.front();

        q.pop();

        cout<<temp->data<<" ";
        if(temp->left){
            q.push(temp->left);
        }
        if(temp->right){
            q.push(temp->right);
        }
    }
}

int main()
{
    Node* root=NULL;
    root=buildTree();

    levelOrderedTraversal(root);
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data :
20
enter the data for left part 20 node
enter the data :
40
enter the data for left part 40 node
enter the data :
60
enter the data for left part 60 node
enter the data :
-1
enter the data for right part 60 node
enter the data :
80
enter the data for left part 80 node
enter the data :
-1
enter the data for right part 80 node
enter the data :
-1
enter the data for right part 40 node
enter the data :
-1
enter the data for right part 20 node
enter the data :
-1
20 40 60 80
```

Q) print tree using level Ordered Traversal

```
#include <iostream>
#include <queue>

using namespace std;

class Node{
    public :
    int data ;
    Node* left;
    Node* right;

    Node(int data){
        this->data=data;
        left=NULL;
        right=NULL;
    }
};

Node* buildTree(){
    int data;
    cout<<"enter the data :"<<endl;
    cin>>data;
    if(data==-1){
        return NULL;
    }

    Node* root=new Node(data);

    cout<<"enter the data for left part "<<data<<" node"<<endl;
    root->left=buildTree();

    cout<<"enter the data for right part "<<data<<" node"<<endl;
    root->right=buildTree();

    return root;
}
```

```
void levelOrderedTraversal(Node* root){
    queue<Node*> q;

    q.push(root);
    q.push(NULL);
    while(!q.empty()){
        Node * temp=q.front();

        q.pop();

        if(temp==NULL){
            cout<<endl;
            if(!q.empty()){
                q.push(NULL);
            }
        }
        else{

            cout<<temp->data<<" ";
            if(temp->left){
            q.push(temp->left);
            }
        if(temp->right){
            q.push(temp->right);
            }
        }

    }
}

int main()
{
    Node* root=NULL;
    root=buildTree();

    levelOrderedTraversal(root);
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data :
20
enter the data for left part 20 node
enter the data :
40
enter the data for left part 40 node
enter the data :
60
enter the data for left part 60 node
enter the data :
-1
enter the data for right part 60 node
enter the data :
80
enter the data for left part 80 node
enter the data :
-1
enter the data for right part 80 node
enter the data :
-1
enter the data for right part 40 node
enter the data :
-1
enter the data for right part 20 node
enter the data :
-1
20
40
60
80
```

Q)find height using tree(code for in order ,pre order ,post order)

```
#include <iostream>
#include <queue>

using namespace std;

class Node{
    public :
    int data ;
    Node* left;
    Node* right;

    Node(int data){
        this->data=data;
        left=NULL;
        right=NULL;
    }
};

Node* buildTree(){
    int data;
    cout<<"enter the data :"<<endl;
    cin>>data;
    if(data==-1){
        return NULL;
    }

    Node* root=new Node(data);

    cout<<"enter the data for left part "<<data<<" node"<<endl;
    root->left=buildTree();

    cout<<"enter the data for right part "<<data<<" node"<<endl;
    root->right=buildTree();

    return root;
}
```

```
void levelOrderedTraversal(Node* root){
    queue<Node*> q;

    q.push(root);
    q.push(NULL);
    while(!q.empty()){
        Node * temp=q.front();

        q.pop();

        if(temp==NULL){
            cout<<endl;
            if(!q.empty()){
                q.push(NULL);
            }
        }
        else{

            cout<<temp->data<<" ";
            if(temp->left){
            q.push(temp->left);
            }
        if(temp->right){
            q.push(temp->right);
            }
        }

    }
}

void inorderTraversal(Node* root){
    if(root==NULL){
        return;
    }
    inorderTraversal(root->left);
    cout<<root->data<<" ";
    inorderTraversal(root->right);
}
```

```
void preorderTraversal(Node* root){
    if(root==NULL){
        return;
    }
    cout<<root->data<<" ";
    inorderTraversal(root->left);

    inorderTraversal(root->right);
}

void postorderTraversal(Node* root){
    if(root==NULL){
        return;
    }

    inorderTraversal(root->left);
    inorderTraversal(root->right);
    cout<<root->data<<" ";
}

int height(Node* root){

    if(root==NULL){
        return 0;
    }

    int leftHeight=height(root->left);
    int rightHeight=height(root->right);
    int ans=max(leftHeight,rightHeight)+1;
    return ans;
}

int main()
{
    Node* root=NULL;
    root=buildTree();

    levelOrderedTraversal(root);
    cout<<"height "<<height(root)<<endl;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data :
10
enter the data for left part 10 node
enter the data :
20
enter the data for left part 20 node
enter the data :
40
enter the data for left part 40 node
enter the data :
-1
enter the data for right part 40 node
enter the data :
-1
enter the data for right part 20 node
enter the data :
50
enter the data for left part 50 node
enter the data :
-1
enter the data for right part 50 node
enter the data :
-1
enter the data for right part 10 node
enter the data :
30
enter the data for left part 30 node
enter the data :
-1
enter the data for right part 30 node
enter the data :
60
```

```
enter the data for left part 60 node
enter the data :
-1
enter the data for right part 60 node
enter the data :
70
enter the data for left part 70 node
enter the data :
-1
enter the data for right part 70 node
enter the data :
-1
10
20 30
40 50 60
70
height 4
```

104. Maximum Depth of Binary Tree

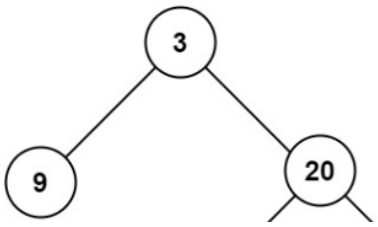
Easy 10.9K 174

Companies

Given the `root` of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:




```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {
 *     TreeNode(int x) : val(x), left(nullptr), right(null
 *     TreeNode(int x, TreeNode *left, TreeNode *right) :
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {

        if(root==NULL){
            return 0;
        }
        int leftHeight=maxDepth(root->left);
        int rightHeight=maxDepth(root->right);
        int ans=max(leftHeight,rightHeight)+1;
        return ans;
    }
};
```

Description Editorial Solutions (4.2K) Submissions

543. Diameter of Binary Tree

Easy 11.5K 718

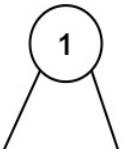
Companies

Given the `root` of a binary tree, return *the length of the **diameter** of the tree*.

The **diameter** of a binary tree is the **length** of the longest path between any two nodes in a tree. This path may or may not pass through the `root`.

The **length** of a path between two nodes is represented by the number of edges between them.

Example 1:



C++ Auto

```cpp
*     TreeNode(int x) : val(x), left(nullptr), right(nullptr)
*     TreeNode(int x, TreeNode *left, TreeNode *right) : val(
* };
*/
class Solution {
public:
    int maxDepth(TreeNode* root){
        if(root==NULL){
            return 0;
        }
        int leftHeight=maxDepth(root->left);
        int rightHeight=maxDepth(root->right);
        int ans=max(leftHeight,rightHeight)+1;
        return ans;
    }
    int diameterOfBinaryTree(TreeNode* root) {
        if(root==NULL){
            return 0;
        }
        int op1=diameterOfBinaryTree(root->left);
        int op2=diameterOfBinaryTree(root->right);
        int op3=maxDepth(root->left)+maxDepth(root->right);
        int ans=max( op1,max(op2,op3));
        return ans;
    }
};
```

Trees Class – 2

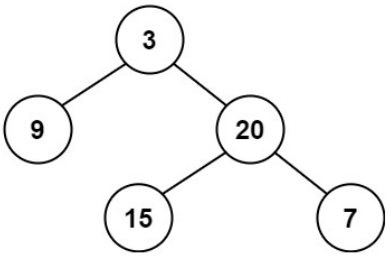
110. Balanced Binary Tree

Easy 9.1K 510

Companies

Given a binary tree, determine if it is **height-balanced**.

Example 1:

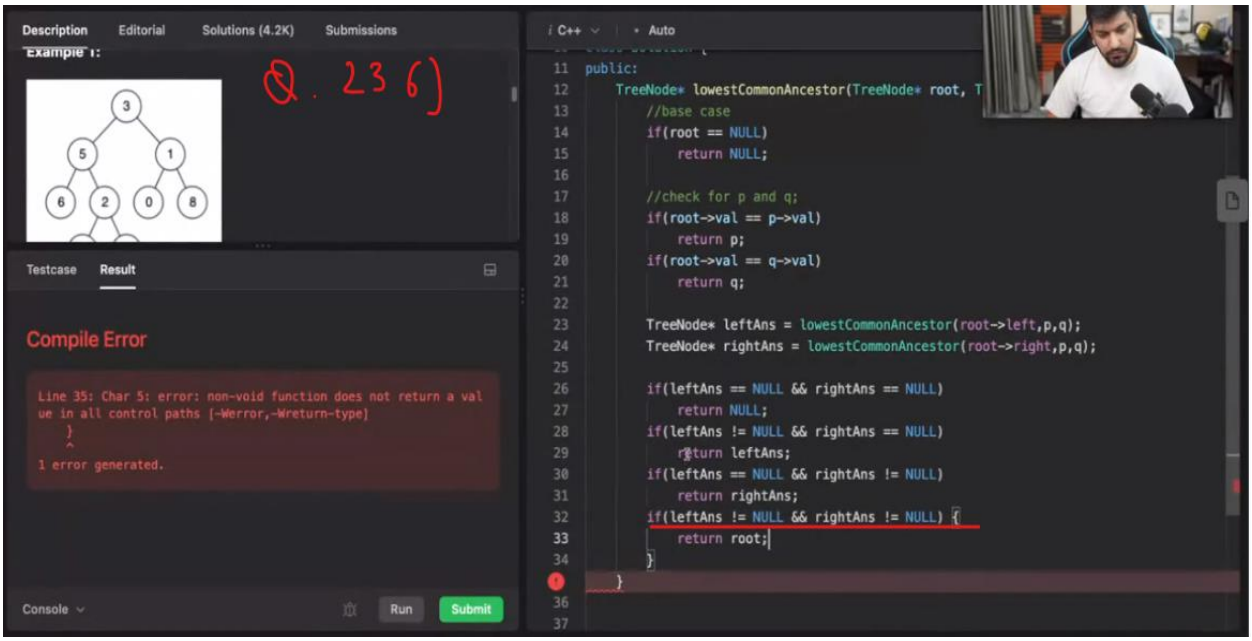


Input: root = [3,9,20,null,null,15,7]

```cpp
public:
    int height(TreeNode* root){
        if(root==NULL){
            return 0;
        }
        int leftHeight = height(root->left);
        int rightHeight = height(root->right);
        int ans = 1 + max(leftHeight , rightHeight);
        return ans;
    }
    bool isBalanced(TreeNode* root) {
        if(root==NULL){
            return true;
        }

        int leftHeight = height(root->left);
        int rightHeight = height(root->right);
        int diff = abs(leftHeight - rightHeight);
        bool ans1 = (diff<=1);

        bool leftAns = isBalanced(root->left);
        bool rightAns = isBalanced(root->right);

        if(ans1 && leftAns && rightAns){
            return true;
```

Console

Q) LeetCode (236)



236. Lowest Common Ancestor of a Binary Tree

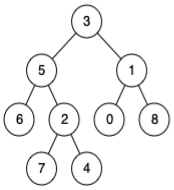
Medium 14.2K 332

Companies

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:




```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root==NULL){
            return NULL;
        }

        if(root->val==p->val){
            return p;
        }
        if(root->val==q->val){
            return q;
        }

        TreeNode* leftAns=lowestCommonAncestor(root->left,p,q);
        TreeNode* rightAns=lowestCommonAncestor(root->right,p,q);

        if(leftAns==NULL && rightAns==NULL)
            return NULL;

        else if(leftAns!=NULL && rightAns==NULL)
            return leftAns;

        else if(leftAns==NULL && rightAns!=NULL)
            return rightAns;
```

Console

Run

Q) K^{th} Ancestor

```
#include <iostream>

using namespace std;

class Node{

    public :
    int data ;
    Node* left;
    Node* right;

    Node(int data){
        this->data=data;
        left=NULL;
        right=NULL;
    }
};

Node* buildTree(){

    int data;
    cout<<"enter the data :"<<endl;
    cin>>data;

    if(data==-1){
        return NULL;
    }

    Node* root=new Node(data);

    cout<<"enter the data for left part "<<data<<" node"<<endl;
    root->left=buildTree();

    cout<<"enter the data for right part "<<data<<" node"<<endl;
    root->right=buildTree();

    return root;
}
```

```cpp
bool kthAncestor(Node *root, int& k, int p)
{
    if(root==NULL){
        return false;
    }

    if(root->data==p){
        return true;
    }
    bool leftAns=kthAncestor(root->left,k,p);
    bool rightAns=kthAncestor(root->right,k,p);

    if(leftAns==true || rightAns==true){
        k--;
    }
    if(k==0){
        cout<<"answer "<<root->data<<endl;
        k=-1;
    }
    return leftAns || rightAns;
}

int main()
{
    Node* root=NULL;
    root=buildTree();

    int k=1;
    int p=4;
    bool found=kthAncestor(root,k,p);
    return 0;
}
```

```
PS E:\C++Code> g++ .\f7.cpp
PS E:\C++Code> .\a.exe
enter the data :
1
enter the data for left part 1 node
enter the data :
2
enter the data for left part 2 node
enter the data :
4
enter the data for left part 4 node
enter the data :
-1
enter the data for right part 4 node
enter the data :
-1
enter the data for right part 2 node
enter the data :
5
enter the data for left part 5 node
enter the data :
-1
enter the data for right part 5 node
enter the data :
-1
enter the data for right part 1 node
enter the data :
3
enter the data for left part 3 node
enter the data :
-1
enter the data for right part 3 node
enter the data :
-1
answer 2
```

Trees Class – 4

Q) Inorder & Preorder Traversal (Imp)

```
#include <iostream>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;

    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }

};

int findPosition(int arr[],int n,int element){
    for(int i=0;i<n;i++){
        if(arr[i]==element){
            return i;
        }
    }
    return -1;
}
```

```
Node* buildTreeFromPreOrderInOrder(int inorder[],int preorder[],int size,int
&preIndex,int inorderStart,int inorderEnd){

    if(preIndex>=size || inorderStart>inorderEnd){
        return NULL;
    }

    int element=preorder[preIndex++];
    Node* root=new Node(element);

    int pos=findPosition(inorder,size,element);

    root-
>left=buildTreeFromPreOrderInOrder(inorder,preorder,size,preIndex,inorderStart,po
s-1);

    root-
>right=buildTreeFromPreOrderInOrder(inorder,preorder,size,preIndex,pos+1,inorderE
nd);

    return root;
}
```

```cpp
void levelOrderTraversal(Node* root ) {
    queue<Node*> q;
    //initially
    q.push(root);
    q.push(NULL);

    while(!q.empty()) {
        //A
        Node* temp = q.front();
        //B
        q.pop();

        if(temp == NULL) {
            cout << endl;
            if(!q.empty()) {
                q.push(NULL);
            }
        }
        else {
            //C
            cout << temp->data << " ";
            //D
            if(temp -> left) {
                q.push(temp ->left);
            }
            if(temp ->right) {
                q.push(temp->right);
            }
        }

    }
}
```

```cpp
int main()
{
    int inorder[] = {40, 20, 50, 10, 60, 30, 70};
    int preorder[] = {10, 20, 40, 50, 30, 60, 70};
    int size=7;
    int preIndex=0;
    int inorderStart=0;
    int inorderEnd=size-1;

    cout<<"building tree :"<<endl;
    Node*
root=buildTreeFromPreOrderInOrder(inorder,preorder,size,preIndex,inorderStart,ino
rderEnd);

    cout<<"printing level order traversal :"<<endl;
    levelOrderTraversal(root);
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
building tree :
printing level order traversal :
10
20 30
40 50 60 70
```

Q) Create a tree using Inorder & Postorder (Imp)

```
#include <iostream>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;

    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }

};

int findPosition(int arr[],int n,int element){
    for(int i=0;i<n;i++){
        if(arr[i]==element){
            return i;
        }
    }
    return -1;
}
```

```
Node* buildTreeFromPostOrderInOrder(int inorder[],int postorder[],int size,int
&postIndex,int inorderStart,int inorderEnd){

    if(postIndex < 0 || inorderStart>inorderEnd){
        return NULL;
    }

    int element=postorder[postIndex];
    postIndex--;
    Node* root=new Node(element);

    int pos=findPosition(inorder,size,element);

    root-
>right=buildTreeFromPostOrderInOrder(inorder,postorder,size,postIndex,pos+1,inord
erEnd);
    root-
>left=buildTreeFromPostOrderInOrder(inorder,postorder,size,postIndex,inorderStart
,pos-1);

    return root;
}
```

```
void levelOrderTraversal(Node* root ) {
    queue<Node*> q;

    q.push(root);
    q.push(NULL);

    while(!q.empty()) {

        Node* temp = q.front();

        q.pop();

        if(temp == NULL) {
            cout << endl;
            if(!q.empty()) {
                q.push(NULL);
            }
        }
        else {

            cout << temp->data << " ";

            if(temp -> left) {
                q.push(temp ->left);
            }
            if(temp ->right) {
                q.push(temp->right);
            }
        }

    }
}


int main()
{
    int inorder[] = {40, 20, 10,50,30,60};
    int postorder[] = {40,20,50,60,30,10};
    int size=6;
    int postOrderIndex=size-1;
    int inorderStart=0;
    int inorderEnd=size-1;
```

```
      cout<<"building tree :"<<endl;

    Node*
root=buildTreeFromPostOrderInOrder(inorder,postorder,size,postOrderIndex,inorderS
tart,inorderEnd);

    cout<<"printing level order traversal :"<<endl;

    levelOrderTraversal(root);
    return 0;
}
```

```
Building the tree:
Printing the tree
10
20 30
40 50 60
```

Q) Create a tree using Inorder & Postorder using mapping (Imp)

```
#include <iostream>
#include <queue>
#include <unordered_map>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;

    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }

};


Node* buildTreeFromPostOrderInOrder(int inorder[],int postorder[],int size,int
&postIndex,int inorderStart,int inorderEnd,
unordered_map<int,int> &mapping){

    if(postIndex < 0 || inorderStart>inorderEnd){
        return NULL;
    }

    int element=postorder[postIndex];
    postIndex--;
    Node* root=new Node(element);

    int pos=mapping[element];
```

```
    root->right=buildTreeFromPostOrderInOrder(inorder,postorder,size,postIndex,pos+1,inorderEnd,mapping);
    root->left=buildTreeFromPostOrderInOrder(inorder,postorder,size,postIndex,inorderStart,pos-1,mapping);

    return root;
}

void levelOrderTraversal(Node* root ) {
    queue<Node*> q;

    q.push(root);
    q.push(NULL);

    while(!q.empty()) {

        Node* temp = q.front();

        q.pop();

        if(temp == NULL) {
            cout << endl;
            if(!q.empty()) {
                q.push(NULL);
            }
        }
        else {

            cout << temp->data << " ";

            if(temp -> left) {
                q.push(temp ->left);
            }
            if(temp ->right) {
                q.push(temp->right);
            }
        }

    }
}

void creatMapping(unordered_map<int,int> &mapping,int inorder[],int size){
    for(int i=0;i<size;i++){
```

```
        mapping[inorder[i]]=i;
    }
}
int main()
{
    int inorder[] = {40, 20, 10,50,30,60};
    int postorder[] = {40,20,50,60,30,10};
    int size=6;
    int postOrderIndex=size-1;
    int inorderStart=0;
    int inorderEnd=size-1;

    unordered_map<int,int> mapping;
    creatMapping(mapping,inorder,size);

    cout<<"building tree :"<<endl;
    Node*
root=buildTreeFromPostOrderInOrder(inorder,postorder,size,postOrderIndex,inorderS
tart,inorderEnd,mapping);

    cout<<"printing level order traversal :"<<endl;
    levelOrderTraversal(root);
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
building tree :
printing level order traversal :
10
20 30
40 50 60
```

Trees Class – 5

Q) write program for top view

```
#include <iostream>
#include <queue>
#include <map>
#include <unordered_map>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;

    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }

};
```

```cpp
Node* buildTree(){

    int data;
    cout<<"enter the data :"<<endl;
    cin>>data;

    if(data==-1){
        return NULL;
    }

    Node* root=new Node(data);

    cout<<"enter the data for left part "<<data<<" node"<<endl;
    root->left=buildTree();

    cout<<"enter the data for right part "<<data<<" node"<<endl;
    root->right=buildTree();

    return root;
}
```

```cpp
void printTopView(Node* root){
    if(root==NULL){
        return;
    }
    map<int,int> topNode;

    queue< pair<Node*,int> >q;

    q.push(make_pair(root,0));
    while(!q.empty()){

        pair<Node*,int>temp=q.front();
        q.pop();

        Node* frontNode=temp.first;
        int hd=temp.second;

        if(topNode.find(hd)==topNode.end()){
            topNode[hd]=frontNode->data;
        }

        if(frontNode->left){
            q.push(make_pair(frontNode->left,hd-1));
        }

          if(frontNode->right){
            q.push(make_pair(frontNode->right,hd+1));
        }

    }
    cout<<"print the top view :"<<endl;
    for(auto i:topNode){
        cout<<i.first<<"->"<<i.second<<endl;
    }
}

int main()
{
    Node* root=buildTree();
    printTopView(root);
    return 0;
    //10,20,30,-1,-1,40,60,-1,-1,-1,80,50,-1,70,-1,-1,90,-1,-1
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data :
10
enter the data for left part 10 node
enter the data :
20
enter the data for left part 20 node
enter the data :
30
enter the data for left part 30 node
enter the data :
-1
enter the data for right part 30 node
enter the data :
-1
enter the data for right part 20 node
enter the data :
40
enter the data for left part 40 node
enter the data :
60
enter the data for left part 60 node
enter the data :
-1
enter the data for right part 60 node
enter the data :
-1
enter the data for right part 40 node
enter the data :
-1
enter the data for right part 10 node
enter the data :
80
```

```
enter the data for left part 80 node
enter the data :
50
enter the data for left part 50 node
enter the data :
-1
enter the data for right part 50 node
enter the data :
70
enter the data for left part 70 node
enter the data :
-1
enter the data for right part 70 node
enter the data :
-1
enter the data for right part 80 node
enter the data :
90
enter the data for left part 90 node
enter the data :
-1
enter the data for right part 90 node
enter the data :
-1
print the top view :
-2->30
-1->20
0->10
1->80
2->90
```

Q) write program for bottom view

```
#include <iostream>
#include <queue>
#include <map>
#include <unordered_map>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;

    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }

};
```

```cpp
Node* buildTree(){

    int data;
    cout<<"enter the data :"<<endl;
    cin>>data;

    if(data==-1){
        return NULL;
    }

    Node* root=new Node(data);

    cout<<"enter the data for left part "<<data<<" node"<<endl;
    root->left=buildTree();

    cout<<"enter the data for right part "<<data<<" node"<<endl;
    root->right=buildTree();

    return root;
}
```

```cpp
void printBottomView(Node* root){
    if(root==NULL){
        return;
    }
    map<int,int> topNode;

    queue< pair<Node*,int> >q;

    q.push(make_pair(root,0));
    while(!q.empty()){

        pair<Node*,int>temp=q.front();
        q.pop();

        Node* frontNode=temp.first;
        int hd=temp.second;


        topNode[hd]=frontNode->data;


        if(frontNode->left){
            q.push(make_pair(frontNode->left,hd-1));
        }

          if(frontNode->right){
            q.push(make_pair(frontNode->right,hd+1));
        }

    }
    cout<<"print the top view :"<<endl;
    for(auto i:topNode){
        cout<<i.first<<"->"<<i.second<<endl;
    }
}

int main()
{
    Node* root=buildTree();
    printBottomView(root);
    return 0;
    //10,20,30,-1,-1,40,60,-1,-1,-1,80,50,-1,70,-1,-1,90,-1,-1
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data :
10 20 30 -1 -1 40 60 -1 -1 -1 80 50 -1 70 -1 -1 90 -1 -1
enter the data for left part 10 node
enter the data :
enter the data for left part 20 node
enter the data :
enter the data for left part 30 node
enter the data :
enter the data for right part 30 node
enter the data :
enter the data for right part 20 node
enter the data :
enter the data for left part 40 node
enter the data :
enter the data for left part 60 node
enter the data :
enter the data for right part 60 node
enter the data :
enter the data for right part 40 node
enter the data :
enter the data for right part 10 node
enter the data :
enter the data for left part 80 node
enter the data :
enter the data for left part 50 node
enter the data :
enter the data for right part 50 node
enter the data :
enter the data for left part 70 node
enter the data :
enter the data for right part 70 node
enter the data :
enter the data for right part 80 node
enter the data :
enter the data for left part 90 node
enter the data :
enter the data for right part 90 node
enter the data :
print the top view :
-2->30
-1->60
0->50
1->70
2->90
```

Q) write program for left view

```
#include <iostream>
#include <queue>
#include <map>
#include <unordered_map>
#include <vector>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;

    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }

};
```

```
Node* buildTree(){

    int data;
    cout<<"enter the data :"<<endl;
    cin>>data;

    if(data==-1){
        return NULL;
    }

    Node* root=new Node(data);

    cout<<"enter the data for left part "<<data<<" node"<<endl;
    root->left=buildTree();

    cout<<"enter the data for right part "<<data<<" node"<<endl;
    root->right=buildTree();

    return root;
}

void printLeftView(Node* root,vector <int> &ans,int level){
    if(root==NULL){
        return;
    }
    if(ans.size()==level){
        ans.push_back(root->data);
    }

    printLeftView(root->left,ans,level+1);
    printLeftView(root->right,ans,level+1);
}
```

```cpp
int main()
{
    Node* root=buildTree();
    vector <int>ans;
    int level=0;
    printLeftView(root,ans,level);

    cout<<"print the left view :"<<endl;
    for(auto i:ans){
        cout<<i<<" ";
    }
    return 0;
    //10,20,30,-1,-1,40,60,-1,-1,-1,80,50,-1,70,-1,-1,90,-1,-1
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data :
10 20 30 -1 -1 40 60 -1 -1 -1 80 50 -1 70 -1 -1 90 -1 -1
enter the data for left part 10 node
enter the data :
enter the data for left part 20 node
enter the data :
enter the data for left part 30 node
enter the data :
enter the data for right part 30 node
enter the data :
enter the data for right part 20 node
enter the data :
enter the data for left part 40 node
enter the data :
enter the data for left part 60 node
enter the data :
enter the data for right part 60 node
enter the data :
enter the data for right part 40 node
enter the data :
enter the data for right part 10 node
enter the data :
enter the data for left part 80 node
enter the data :
enter the data for left part 50 node
enter the data :
enter the data for right part 50 node
enter the data :
enter the data for left part 70 node
enter the data :
enter the data for right part 70 node
enter the data :
enter the data for right part 80 node
enter the data :
90
enter the data for left part 90 node
enter the data :
-1
enter the data for right part 90 node
enter the data :
-1
print the left view :
10 20 30 60
```

Q) write program for right view

```
#include <iostream>
#include <queue>
#include <map>
#include <unordered_map>
#include <vector>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;

    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }

};
```

```
Node* buildTree(){

    int data;
    cout<<"enter the data :"<<endl;
    cin>>data;

    if(data==-1){
        return NULL;
    }

    Node* root=new Node(data);

    cout<<"enter the data for left part "<<data<<" node"<<endl;
    root->left=buildTree();

    cout<<"enter the data for right part "<<data<<" node"<<endl;
    root->right=buildTree();

    return root;
}

void printRightView(Node* root,vector <int> &ans,int level){
    if(root==NULL){
        return;
    }
    if(ans.size()==level){
        ans.push_back(root->data);
    }

    printRightView(root->right,ans,level+1);
    printRightView(root->left,ans,level+1);

}
```

```cpp
int main()
{
    Node* root=buildTree();
    vector <int>ans;
    int level=0;
    printRightView(root,ans,level);

    cout<<"print the left view :"<<endl;
    for(auto i:ans){
        cout<<i<<" ";
    }
    return 0;
    //10,20,30,-1,-1,40,60,-1,-1,-1,80,50,-1,70,-1,-1,90,-1,-1
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data :
10 20 30 -1 -1 40 60 -1 -1 -1 80 50 -1 70 -1 -1 90 -1 -1
enter the data for left part 10 node
enter the data :
enter the data for left part 20 node
enter the data :
enter the data for left part 30 node
enter the data :
enter the data for right part 30 node
enter the data :
enter the data for right part 20 node
enter the data :
enter the data for left part 40 node
enter the data :
enter the data for left part 60 node
enter the data :
enter the data for right part 60 node
enter the data :
enter the data for right part 40 node
enter the data :
enter the data for right part 10 node
enter the data :
enter the data for left part 80 node
enter the data :
enter the data for left part 50 node
enter the data :
enter the data for right part 50 node
enter the data :
enter the data for left part 70 node
enter the data :
enter the data for right part 70 node
enter the data :
enter the data for right part 80 node
enter the data :
enter the data for left part 90 node
```

```
enter the data :
enter the data for right part 90 node
enter the data :
print the right view :
10 80 90 70
```

Q) Boundary Traversal View

```cpp
#include <iostream>
#include <queue>
#include <map>
#include <unordered_map>
#include <vector>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;

    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }

};
```

```
Node* buildTree(){

    int data;
    cout<<"enter the data :"<<endl;
    cin>>data;

    if(data==-1){
        return NULL;
    }

    Node* root=new Node(data);

    cout<<"enter the data for left part "<<data<<" node"<<endl;
    root->left=buildTree();

    cout<<"enter the data for right part "<<data<<" node"<<endl;
    root->right=buildTree();

    return root;
}

void printLeftBoundary(Node* root){
    if(root==NULL){
        return;
    }
    if(root->left==NULL && root->right==NULL){
        return;
    }
    cout<<root->data<<" ";

    if(root->left){
        printLeftBoundary(root->left);
    }
    else{
        printLeftBoundary(root->right);
    }
}
```

```cpp
void printLeafBoundary(Node* root){
    if(root==NULL){
        return;
    }
    if(root->left==NULL && root->right==NULL){
        cout<<root->data<<" ";
    }
    printLeafBoundary(root->left);
    printLeafBoundary(root->right);
}

void printRightBoundary(Node* root){
    if(root==NULL){
        return ;
    }
    if(root->left==NULL && root->right==NULL){
        return;
    }

    if(root->right){
        printRightBoundary(root->right);
    }
    else{
        printRightBoundary(root->left);
    }
    cout<<root->data<<" ";
}

void boundaryTraversal(Node* root){
    if(root==NULL){
        return;
    }
    cout<<root->data<<" ";

    printLeftBoundary(root->left);

    printLeafBoundary(root);

    printRightBoundary(root->right);
}
```

```cpp
int main()
{
    Node* root=buildTree();
    boundaryTraversal(root);
    return 0;
    //10,20,30,-1,-1,40,60,-1,-1,-1,80,50,-1,70,-1,-1,90,-1,-1
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data :
10 20 30 -1 -1 40 60 -1 -1 -1 80 50 -1 70 -1 -1 90 -1 -1
enter the data for left part 10 node
enter the data :
enter the data for left part 20 node
enter the data :
enter the data for left part 30 node
enter the data :
enter the data for right part 30 node
enter the data :
enter the data for right part 20 node
enter the data :
enter the data for left part 40 node
enter the data :
enter the data for left part 60 node
enter the data :
enter the data for right part 60 node
enter the data :
enter the data for right part 40 node
enter the data :
enter the data for right part 10 node
enter the data :
enter the data for left part 80 node
enter the data :
enter the data for left part 50 node
enter the data :
enter the data for right part 50 node
enter the data :
enter the data for left part 70 node
enter the data :
enter the data for right part 70 node
enter the data :
enter the data for right part 80 node
enter the data :
enter the data for left part 90 node
```

```
enter the data :
enter the data for right part 90 node
enter the data :
boundary tarversal view :
10 20 30 60 70 90 80
```

Binary Search Trees

BST Class -1

Q) write program for Binary Search Tree

```
#include <iostream>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;
    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }
};

Node * insertIntoBST(Node* root,int data){

    if(root==NULL){
        root=new Node(data);
        return root;
    }

    if(root->data > data){
        root->left=insertIntoBST(root->left,data);
    }
    else{
        root->right=insertIntoBST(root->right,data);
    }
    return root;
}
```

```
void takeInput(Node* &root){
    int data;
    cout<<"enter data :"<<endl;
    cin>>data;

    while(data!=-1){
        root=insertIntoBST(root,data);
        cout<<"enter data :"<<endl;
        cin>>data;
    }
}

void levelOrderTraversal(Node* root ) {
    queue<Node*> q;

    q.push(root);
    q.push(NULL);

    while(!q.empty()) {

        Node* temp = q.front();

        q.pop();

        if(temp == NULL) {
            cout << endl;
            if(!q.empty()) {
                q.push(NULL);
            }
        }
        else {

            cout << temp->data << " ";

            if(temp -> left) {
                q.push(temp ->left);
            }
            if(temp ->right) {
                q.push(temp->right);
            }
        }

    }
}
int main()
```

```cpp
{
    Node* root=NULL;
    cout<<"enter the data for node :"<<endl;
    takeInput(root);
    cout<<"printing the tree :"<<endl;
    levelOrderTraversal(root);
    return 0;

}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data for node :
enter data :
10 20 5 11 17 2 4 8 6 25 15 -1
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
printing the tree :
10
5 20
2 8 11 25
4 6 17
15
```

Q) write program for Binary Search Tree with inOrder ,preorder & postOrder

```
#include <iostream>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;

    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }

};

Node * insertIntoBST(Node* root,int data){

    if(root==NULL){
        root=new Node(data);
        return root;
    }

    if(root->data > data){
        root->left=insertIntoBST(root->left,data);
    }
    else{
        root->right=insertIntoBST(root->right,data);
    }

    return root;
}
```

```
void takeInput(Node* &root){
    int data;
    cout<<"enter data :"<<endl;
    cin>>data;

    while(data!=-1){
        root=insertIntoBST(root,data);
        cout<<"enter data :"<<endl;
        cin>>data;
    }
}

void levelOrderTraversal(Node* root ) {
    queue<Node*> q;

    q.push(root);
    q.push(NULL);

    while(!q.empty()) {

        Node* temp = q.front();

        q.pop();

        if(temp == NULL) {
            cout << endl;
            if(!q.empty()) {
                q.push(NULL);
            }
        }
        else {

            cout << temp->data << " ";

            if(temp -> left) {
                q.push(temp ->left);
            }
            if(temp ->right) {
                q.push(temp->right);
            }
        }

    }
}
```

```
void inOrderTraversal(Node* root){
    if(root==NULL){
        return;
    }
    inOrderTraversal(root->left);
    cout<<root->data<<" ";
    inOrderTraversal(root->right);
}
void preOrderTraversal(Node* root){
    if(root==NULL){
        return;
    }

    cout<<root->data<<" ";
    preOrderTraversal(root->left);
    preOrderTraversal(root->right);
}

void postOrderTraversal(Node* root){
    if(root==NULL){
        return;
    }

    postOrderTraversal(root->left);
    postOrderTraversal(root->right);
    cout<<root->data<<" ";

}

int main()
{
    Node* root=NULL;
    cout<<"enter the data for node :"<<endl;
    takeInput(root);

    cout<<"printing the tree :"<<endl;
    levelOrderTraversal(root);

    cout<<endl;
    cout<<"inOrder tervarsal :";
    inOrderTraversal(root);

    cout<<endl;
    cout<<"preOrder tervarsal :";
    preOrderTraversal(root);
```

```
    cout<<endl;
    cout<<"postOrder tervarsal :";
    postOrderTraversal(root);
    //10 20 5 11 17 2 4 8 6 25 15 -1
    return 0;

}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data for node :
enter data :
10 20 5 11 17 2 4 8 6 25 15 -1
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
printing the tree :
10
5 20
2 8 11 25
4 6 17
15

inOrder tervarsal :2 4 5 6 8 10 11 15 17 20 25
preOrder tervarsal :10 5 2 4 8 6 20 11 17 15 25
postOrder tervarsal :4 2 6 8 5 15 17 11 25 20 10
```

Q) Find element in BST

```
#include <iostream>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;

    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }

};

Node * insertIntoBST(Node* root,int data){

    if(root==NULL){
        root=new Node(data);
        return root;
    }

    if(root->data > data){
        root->left=insertIntoBST(root->left,data);
    }
    else{
        root->right=insertIntoBST(root->right,data);
    }

    return root;
}
```

```
void takeInput(Node* &root){
    int data;
    cout<<"enter data :"<<endl;
    cin>>data;

    while(data!=-1){
        root=insertIntoBST(root,data);
        cout<<"enter data :"<<endl;
        cin>>data;
    }
}

void levelOrderTraversal(Node* root ) {
    queue<Node*> q;

    q.push(root);
    q.push(NULL);

    while(!q.empty()) {

        Node* temp = q.front();

        q.pop();

        if(temp == NULL) {
            cout << endl;
            if(!q.empty()) {
                q.push(NULL);
            }
        }
        else {

            cout << temp->data << " ";

            if(temp -> left) {
                q.push(temp ->left);
            }
            if(temp ->right) {
                q.push(temp->right);
            }
        }

    }
}
```

```cpp
bool findNodeInBST(Node* root, int target){
  if(root == NULL){
    return false;
  }
  if(root->data==target){
    return true;
  }

  if(target > root->data){
    return findNodeInBST(root->right,target);
  }
  else{
    return findNodeInBST(root->left,target);
  }
}

int main()
{
    Node* root=NULL;
    cout<<"enter the data for node :"<<endl;
    takeInput(root);

    cout<<"printing the tree :"<<endl;
    levelOrderTraversal(root);

    int target=15;
    bool find=findNodeInBST(root,target);
    if(find){
      cout<<target<<" is present in BST "<<endl;
    }
    else{
      cout<<target<<" is NOT present in BST "<<endl;
    }
    //10 20 5 11 17 2 4 8 6 25 15 -1
    return 0;

}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data for node :
enter data :
10 20 5 11 17 2 4 8 6 25 15 -1
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
printing the tree :
10
5 20
2 8 11 25
4 6 17
15
15 is present in BST
```

Q) write program for find min & max value in BST

```
#include <iostream>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node* left;
    Node* right;

    Node()
    {
        this->data = 0;
        this->left = NULL;
        this->right=NULL;
    }
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right=NULL;
    }

};

Node * insertIntoBST(Node* root,int data){

    if(root==NULL){
        root=new Node(data);
        return root;
    }

    if(root->data > data){
        root->left=insertIntoBST(root->left,data);
    }
    else{
        root->right=insertIntoBST(root->right,data);
    }

    return root;
}
```

```
void takeInput(Node* &root){
    int data;
    cout<<"enter data :"<<endl;
    cin>>data;

    while(data!=-1){
        root=insertIntoBST(root,data);
        cout<<"enter data :"<<endl;
        cin>>data;
    }
}

void levelOrderTraversal(Node* root ) {
    queue<Node*> q;

    q.push(root);
    q.push(NULL);

    while(!q.empty()) {

        Node* temp = q.front();

        q.pop();

        if(temp == NULL) {
            cout << endl;
            if(!q.empty()) {
                q.push(NULL);
            }
        }
        else {

            cout << temp->data << " ";

            if(temp -> left) {
                q.push(temp ->left);
            }
            if(temp ->right) {
                q.push(temp->right);
            }
        }

    }
}
```

```
int minValue(Node* root){

    Node* temp=root;
    if(temp==NULL){
        return -1;
    }

    while(temp->left!=NULL){
        temp=temp->left;
    }

    return temp->data;
}

int maxValue(Node* root){

    Node* temp=root;

    if(temp==NULL){
        return -1;
    }

    while(temp->right!=NULL){
        temp=temp->right;
    }

    return temp->data;
}
int main()
{
    Node* root=NULL;
    cout<<"enter the data for node :"<<endl;
    takeInput(root);

    cout<<"printing the tree :"<<endl;
    levelOrderTraversal(root);

    int minimumValue =minValue(root);
    cout<<"minimum Value :"<<minimumValue<<endl;

    int maximumValue =maxValue(root);
    cout<<"maximum Value :"<<maximumValue<<endl;
    //10 20 5 11 17 2 4 8 6 25 15 -1
    return 0;
    }
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the data for node :
enter data :
10 20 5 11 17 2 4 8 6 25 15 -1
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
enter data :
printing the tree :
10
5 20
2 8 11 25
4 6 17
15
minimum Value :2
maximum Value :25
```

Q) 450. Delete Node in a BST

Description | Editorial | Solutions (2.6K) | Submissions

450. Delete Node in a BST

Medium | 7.7K | 200

Companies

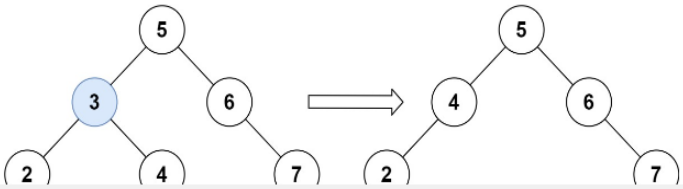
Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return *the **root node reference** (possibly updated) of the BST*.

Basically, the deletion can be divided into two stages:

Search for a node to remove.

If the node is found, delete the node.

Example 1:



C++ | Auto

```
class Solution {
public:
    int maxVal(TreeNode *root)
    {
        TreeNode *temp = root;
        if (temp == NULL)
        {
            return -1;
        }

        while (temp->right != NULL)
        {
            temp = temp->right;
        }
        return temp->val;
    }
    TreeNode* deleteNode(TreeNode* root, int key) {
        if (root == NULL)
    {
        return NULL;
    }

    if (root->val == key)
    {
        if (root->left == NULL && root->right == NULL)
        {
```

Console

```cpp
#include <iostream>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node *left;
    Node *right;

    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right = NULL;
    }
};

Node *insertIntoBST(Node *root, int data)
{

    if (root == NULL)
    {
        // this is the first node we have to create
        root = new Node(data);
        return root;
    }

    // no the first node

    if (root->data > data)
    {
        // insert in left
        root->left = insertIntoBST(root->left, data);
    }
    else
    {
        // insert into right
        root->right = insertIntoBST(root->right, data);
    }
    return root;
}
```

```
void takeInput(Node *&root)
{
    int data;
    cin >> data;

    while (data != -1)
    {
        root = insertIntoBST(root, data);
        cin >> data;
    }
}

void levelOrderTraversal(Node *root)
{
    queue<Node *> q;
    // initially
    q.push(root);
    q.push(NULL);

    while (!q.empty())
    {
        // A
        Node *temp = q.front();
        // B
        q.pop();

        if (temp == NULL)
        {
            cout << endl;
            if (!q.empty())
            {
                q.push(NULL);
            }
        }
        else
        {
            // C
            cout << temp->data << " ";
            // D
            if (temp->left)
            {
                q.push(temp->left);
            }
```

```
            if (temp->right)
            {
                q.push(temp->right);
            }
        }
    }
}

int maxVal(Node *root)
{
    Node *temp = root;
    if (temp == NULL)
    {
        return -1;
    }

    while (temp->right != NULL)
    {
        temp = temp->right;
    }
    return temp->data;
}

Node *deleteNodeInBST(Node *root, int target)
{

    // base cae
    if (root == NULL)
    {
        return NULL;
    }
    // cout << "Request recieved for " >> root->data << " with target" << target
<< endl;
    if (root->data == target)
    {
        // isi ko delete krna h
        // 4 cases
        if (root->left == NULL && root->right == NULL)
        {
            // leaf node
            // delete root;
            return NULL;
        }
```

```
        else if (root->left == NULL && root->right != NULL)
        {
            Node *child = root->right;
            // delete root;
            return child;
        }
        else if (root->left != NULL && root->right == NULL)
        {
            Node *child = root->left;
            // delete root;
            return child;
        }
        else
        {
            // both child
            // find inorder predecessor inb left subtree
            int inorderPre = maxVal(root->left);
            // replace root->data value with inorder predecessor
            root->data = inorderPre;
            // delete inorder predecessor from left subtree
            root->left = deleteNodeInBST(root->left, inorderPre);
            return root;
        }
    }
    else if (target > root->data)
    {
        // right jana chahiye
        root->right = deleteNodeInBST(root->right, target);
    }
    else if (target < root->data)
    {
        // left jana chahioye
        root->left = deleteNodeInBST(root->left, target);
    }
    return root;
}
int main()
{
    Node *root = NULL;
    cout << "Enter the data for Node " << endl;
    takeInput(root);
    cout << "Printing the tree" << endl;
    levelOrderTraversal(root);
    cout << endl;
```

```
    root = deleteNodeInBST(root, 100);
    levelOrderTraversal(root);
    //100 50 150 40 60 175 110
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
Enter the data for Node
100 50 150 40 60 175 110 -1
Printing the tree
100
50 150
40 60 110 175

60
50 150
40 110 175
```

BST Class -2

98. Validate Binary Search Tree

Medium 14.9K 1.2K

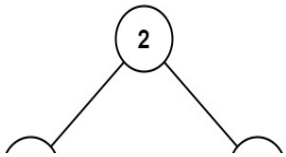
Companies

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST).*

A **valid BST** is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:




```
*/
class Solution {
public:
    bool solve(TreeNode* root,long long int lowerBound,long long int upperBound){
        if(root==NULL){
            return NULL;
        }
        if(root->val > lowerBound && root->val < upperBound ){
            bool leftAns=solve(root->left,lowerBound,root->val);
            bool rightAns=solve(root->right,root->val,upperBound);
            return leftAns && rightAns;
        }
        else{
            return false;
        }
    }
    bool isValidBST(TreeNode* root) {
```

Testcase Result

Wrong Answer Runtime: 0 ms Diff

Case 1 Case 2

Console Run Submit

---

235. Lowest Common Ancestor of a Binary Search Tree

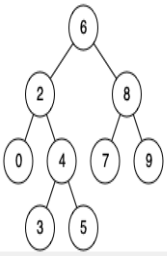
Medium 9.6K 271

Companies

Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two given nodes in the BST.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:




```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */

class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root==NULL){
            return NULL;
        }
        if(p->val < root->val && q->val < root->val){
            return lowestCommonAncestor(root->left,p,q);
        }
        if(p->val > root->val && q->val > root->val){
            return lowestCommonAncestor(root->right,p,q);
        }
        return root;
    }
};
```

Console Run Subm

Tc=O(log n)

Description | Editorial | Solutions (5.9K) | Submissions

230. Kth Smallest Element in a BST

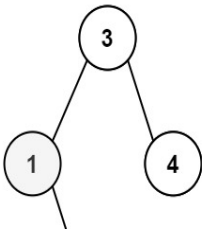
Hint

Medium | 10K | 179

Companies

Given the `root` of a binary search tree, and an integer `k`, return *the* `k^th` *smallest value (**1-indexed**) of all the values of the nodes in the tree.*

Example 1:



C++ | Auto

```
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right
(right) {}
 * };
 */
class Solution {
public:
    int kthSmallest(TreeNode* root, int &k) {
        if(root==NULL){
            return -1;
        }
        int leftAns=kthSmallest(root->left,k);

        if(leftAns!=-1){
            return leftAns;
        }

        k--;
        if(k==0){
            return root->val;
        }

        int rightAns=kthSmallest(root->right,k);
        return rightAns;
    }
};
```

Q) make binary search tree using Inorder

```
#include <iostream>
#include <queue>

using namespace std;

class Node
{
public:

    int data;
    Node *left;
    Node *right;

    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right = NULL;
    }
};
```

```
void levelOrderTraversal(Node *root)
{
    queue<Node *> q;
    // initially
    q.push(root);
    q.push(NULL);

    while (!q.empty())
    {
        // A
        Node *temp = q.front();
        // B
        q.pop();

        if (temp == NULL)
        {
            cout << endl;
            if (!q.empty())
            {
                q.push(NULL);
            }
        }

        else
        {
            // C
            cout << temp->data << " ";
            // D
            if (temp->left)
            {
                q.push(temp->left);
            }


            if (temp->right)
            {
                q.push(temp->right);
            }
        }
    }
}
```

```cpp
Node * bstUsingInorder(int inorder[],int s ,int e){
    if(s>e){
        return NULL;
    }
    int mid=(s+e)/2;
    int element=inorder[mid];
    Node* root=new Node(element);

    root->left=bstUsingInorder(inorder,s,mid-1);
    root->right=bstUsingInorder(inorder,mid+1,e);
    return root;
}

int main()
{
    int inorder[]={1,2,3,4,5,6,7,8,9};
    int s=0;
    int e=8;

    Node* root=bstUsingInorder(inorder,s,e);
    levelOrderTraversal(root);

    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
5
2 7
1 3 6 8
4 9
```

Tc=O(n)

Description | Editorial | Solutions (3.2K) | Submissions

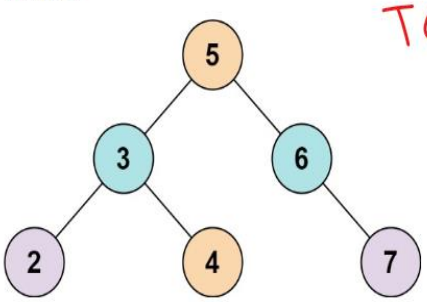
653. Two Sum IV - Input is a BST

Easy | 6K | 243

Companies

Given the `root` of a binary search tree and an integer `k`, return `true` *if there exist two elements in the BST such that their sum is equal to* `k`, *or* `false` *otherwise.*

Example 1:



TC: O(n)

C++ | Auto

```
*/
class Solution {
public:
    void storeInorder(TreeNode* root,vector<int> &inorder){
        if(root==NULL){
            return;
        }
        storeInorder(root->left,inorder);
        inorder.push_back(root->val);
        storeInorder(root->right,inorder);
    }
    bool findTarget(TreeNode* root, int k) {
        vector<int> inorder;
        storeInorder(root,inorder);

        int s=0;
        int e=inorder.size()-1;

        while(s<e){
            int sum=inorder[s]+inorder[e];
            if(sum==k){
                return true;
            }

            if(sum>k){
                e--;
```

Console

BST Class -3

Q) convert BST into sorted linked list

```
#include <iostream>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node *left;
    Node *right;

    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right = NULL;
    }
};

void levelOrderTraversal(Node *root)
{
    queue<Node *> q;
    // initially
    q.push(root);
    q.push(NULL);

    while (!q.empty())
    {
        // A
        Node *temp = q.front();
        // B
        q.pop();

        if (temp == NULL)
        {
            cout << endl;
            if (!q.empty())
            {
                q.push(NULL);
            }
        }
```

```
        else
        {
            // C
            cout << temp->data << " ";
            // D
            if (temp->left)
            {
                q.push(temp->left);
            }


            if (temp->right)
            {
                q.push(temp->right);
            }
        }
    }
}
Node * bstUsingInorder(int inorder[],int s ,int e){
    if(s>e){
        return NULL;
    }
    int mid=(s+e)/2;
    int element=inorder[mid];
    Node* root=new Node(element);

    root->left=bstUsingInorder(inorder,s,mid-1);
    root->right=bstUsingInorder(inorder,mid+1,e);
    return root;
}
void convertedIntoSortedDLL(Node* root,Node* &head){
    if(root==NULL){
        return;
    }
    convertedIntoSortedDLL(root->right,head);
    root->right=head;

    if(head!=NULL){
        head->left=root;
    }

    head=root;
    convertedIntoSortedDLL(root->left,head);
}
```

```cpp
void printingLinkedList(Node* head){
    Node* temp=head;
    while(temp!=NULL){
        cout<<temp->data<<" ";
        temp=temp->right;
    }
    cout<<endl;
}
int main()
{
    int inorder[]={1,2,3,4,5,6,7,8,9};
    int s=0;
    int e=8;

    Node* root=bstUsingInorder(inorder,s,e);
    levelOrderTraversal(root);

    cout<<"printing sorted linked list :"<<endl;
    Node* head=NULL;
    convertedIntoSortedDLL(root,head);
    printingLinkedList(head);
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
5
2 7
1 3 6 8
4 9
printing sorted linked list :
1 2 3 4 5 6 7 8 9
```

Q) convert sorted linked list into BST

```
#include <iostream>
#include <queue>

using namespace std;

class Node
{
public:
    int data;
    Node *left;
    Node *right;

    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right = NULL;
    }
};

void levelOrderTraversal(Node *root)
{
    queue<Node *> q;
    // initially
    q.push(root);
    q.push(NULL);

    while (!q.empty())
    {
        // A
        Node *temp = q.front();
        // B
        q.pop();

        if (temp == NULL)
        {
            cout << endl;
            if (!q.empty())
            {
                q.push(NULL);
            }
        }
        else
```

```
        {
            // C
            cout << temp->data << " ";
            // D
            if (temp->left)
            {
                q.push(temp->left);
            }


            if (temp->right)
            {
                q.push(temp->right);
            }
        }
    }
}

Node * bstUsingInorder(int inorder[],int s ,int e){
    if(s>e){
        return NULL;
    }
    int mid=(s+e)/2;
    int element=inorder[mid];
    Node* root=new Node(element);

    root->left=bstUsingInorder(inorder,s,mid-1);
    root->right=bstUsingInorder(inorder,mid+1,e);
    return root;
}

void convertedIntoSortedDLL(Node* root,Node* &head){
    if(root==NULL){
        return;
    }
    convertedIntoSortedDLL(root->right,head);
    root->right=head;

    if(head!=NULL){
        head->left=root;
    }

    head=root;
    convertedIntoSortedDLL(root->left,head);
}
```

```
void printingLinkedList(Node* head){
    Node* temp=head;
    while(temp!=NULL){
        cout<<temp->data<<" ";
        temp=temp->right;
    }
    cout<<endl;
}
Node* sortedLinkedListIntoBST(Node* &head,int n){
    if(n<=0 || head==NULL){
        return NULL;
    }

    Node* leftSubTree=sortedLinkedListIntoBST(head,n-1-n/2);
    Node* root=head;
    root->left=leftSubTree;

    head=head->right;

    root->right=sortedLinkedListIntoBST(head,n/2);
    return root;
}
int main()
{
    int inorder[]={1,2,3,4,5,6,7,8,9};
    int s=0;
    int e=8;

    Node* root=bstUsingInorder(inorder,s,e);
    levelOrderTraversal(root);

    cout<<"printing sorted linked list :"<<endl;
    Node* head=NULL;
    convertedIntoSortedDLL(root,head);
    printingLinkedList(head);

    cout<<endl;
    cout<<"convert sorted linked into BST "<<endl;
    Node* root1=NULL;
    root1=sortedLinkedListIntoBST(head,9);
    levelOrderTraversal(root1);
    return 0;
}

//------------------------------------------------------------------------//
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
5
2 7
1 3 6 8
4 9
printing sorted linked list :
1 2 3 4 5 6 7 8 9

convert sorted linked into BST
5
2 7
1 3 6 8
4 9
```

Q) find largest BST in a Binary Tree

```
#include <iostream>
#include <queue>
#include <limits.h>

using namespace std;

class Node
{
public:
    int data;
    Node *left;
    Node *right;

    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right = NULL;
    }
};

class NodeData
{
public:
    int size;
    int minVal;
    int maxVal;
    bool validBST;

    NodeData()
    {
    }

    NodeData(int size, int max, int min, bool valid)
    {
        this->size = size;
        maxVal = max;
        minVal = min;
        validBST = valid;
    }
};
```

```cpp
void levelOrderTraversal(Node *root)
{
    queue<Node *> q;
    // initially
    q.push(root);
    q.push(NULL);

    while (!q.empty())
    {
        // A
        Node *temp = q.front();
        // B
        q.pop();

        if (temp == NULL)
        {
            cout << endl;
            if (!q.empty())
            {
                q.push(NULL);
            }
        }
        else
        {
            // C
            cout << temp->data << " ";
            // D
            if (temp->left)
            {
                q.push(temp->left);
            }

            if (temp->right)
            {
                q.push(temp->right);
            }
        }
    }
}
```

```
NodeData *findLargestBST(Node *root, int &ans)
{
    if (root == NULL)
    {
        NodeData *temp = new NodeData(0, INT_MIN, INT_MAX, true);
        return temp;
    }
    NodeData *leftKaAns = findLargestBST(root->left, ans);
    NodeData *rightKaAns = findLargestBST(root->right, ans);

    NodeData *currNodeKaAns = new NodeData();

    currNodeKaAns->size = leftKaAns->size + rightKaAns->size + 1;
    currNodeKaAns->maxVal = max(root->data, rightKaAns->maxVal);
    currNodeKaAns->minVal = min(root->data, leftKaAns->minVal);

    if (leftKaAns->validBST && rightKaAns->validBST && (root->data > leftKaAns-
>maxVal && root->data < rightKaAns->minVal))
    {
        currNodeKaAns->validBST = true;
    }
    else
    {
        currNodeKaAns->validBST = false;
    }

    if (currNodeKaAns->validBST)
    {
        ans = max(ans, currNodeKaAns->size);
    }
    return currNodeKaAns;
}
int main()
{

    Node *root = new Node(50);
    Node *first = new Node(30);
    Node *second = new Node(60);
    Node *third = new Node(5);
    Node *fourth = new Node(20);
    Node *fifth = new Node(45);
    Node *sixth = new Node(70);
    Node *seventh = new Node(65);
    Node *eight = new Node(80);
```

```
    root->left = first;
    root->right = second;
    first->left = third;
    first->right = fourth;
    second->left = fifth;
    second->right = sixth;
    sixth->left = seventh;
    sixth->right = eight;

    cout << "printing the tree " << endl;
    levelOrderTraversal(root);

    int ans = 0;
    findLargestBST(root, ans);
    cout << "largest bst size :" << ans;
    return 0;
}

//-------------------------------------------------------------------------//
```

```
PS E:\C++Code> .\a.exe
printing the tree
50
30 60
5 20 45 70
65 80
largest bst size :5
```

Heaps

Heaps Class -1

Q) insert value in HEAP

```cpp
#include <iostream>

using namespace std;

class Heap{
    public :
        int arr[101];
        int size;

    Heap(){
        size=0;
    }
    void insert (int value){
        size=size+1;
        int index=size;
        arr[index]=value;

        while(index>1){
            int parentIndex=index/2;
            if(arr[index]>arr[parentIndex]){
                swap(arr[index],arr[parentIndex]);
                index=parentIndex;
            }
            else{
                break;
            }
        }
    }
};
int main()
{
    Heap h;
    h.arr[0]=-1;
    h.arr[1]=100;
    h.arr[2]=50;
    h.arr[3]=60;
    h.arr[4]=40;
    h.arr[5]=45;
    h.size=5;
```

```
    cout<<"printig the heap :"<<endl;
    for(int i=0;i<=h.size;i++){
        cout<<h.arr[i]<<" ";
    }
    cout<<endl;

    h.insert(110);
    cout<<endl;
    cout<<"after insert value printig the heap :"<<endl;
    for(int i=0;i<=h.size;i++){
        cout<<h.arr[i]<<" ";
    }
    cout<<endl;

    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
printig the heap :
-1 100 50 60 40 45

after insert value printig the heap :
-1 110 50 100 40 45 60
```

Tc=O(n)

Q)delete element is heap

```cpp
#include <iostream>

using namespace std;

class Heap{
    public :
        int arr[101];
        int size;

    Heap(){
        size=0;
    }
    void insert (int value){
        size=size+1;
        int index=size;
        arr[index]=value;

        while(index>1){
            int parentIndex=index/2;
            if(arr[index]>arr[parentIndex]){
                swap(arr[index],arr[parentIndex]);
                index=parentIndex;
            }
            else{
                break;
            }
        }
    }
```

```
    int deleteMethod(){
        int ans =arr[1];

        arr[1]=arr[size];
        size--;

        int index=1;
        while(index<size){
            int left=2*index;
            int right=2*index+1;

            int largest=index;

            if(left < size && arr[largest] < arr[left]){
                largest=left;
            }

            if(right < size && arr[largest] < arr[right]){
                largest=right;
            }

            if(largest==index){
                break ;
            }else{
                swap(arr[index],arr[largest]);
                index=largest;
            }
        }
        return ans;
    }
};
int main()
{
    Heap h;
    h.arr[0]=-1;
    h.arr[1]=100;
    h.arr[2]=50;
    h.arr[3]=60;
    h.arr[4]=40;
    h.arr[5]=45;
    h.size=5;

    cout<<"printing the heap :"<<endl;
    for(int i=0;i<=h.size;i++){
        cout<<h.arr[i]<<" ";
```

```
    }
    cout<<endl;

    h.insert(110);
    cout<<endl;
    cout<<"after insert value printing the heap :"<<endl;
    for(int i=0;i<=h.size;i++){
        cout<<h.arr[i]<<" ";
    }
    cout<<endl;




    cout<<endl;
    int ans=h.deleteMethod();
    cout<<"delete elements is "<<ans<<endl;
    cout<<"after delete printing the heap :"<<endl;
    for(int i=0;i<=h.size;i++){
        cout<<h.arr[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
printing the heap :
-1 100 50 60 40 45

after insert value printing the heap :
-1 110 50 100 40 45 60

delete elements is 110
after delete printing the heap :
-1 100 50 60 40 45
```

Tc=O(logn)

Q) write program for build heap

```cpp
#include <iostream>

using namespace std;

void heapify(int arr[],int n,int i){
    int index=i;
    int leftIndex=2*i;
    int rightIndex=2*i+1;

    int larget =index;

    if(leftIndex <=n && arr[larget] < arr[leftIndex]){
        larget=leftIndex;
    }
    if(rightIndex <=n && arr[larget] < arr[rightIndex]){
        larget=rightIndex;
    }

    if(index!=larget){
        swap(arr[index],arr[larget]);
        index=larget;
        heapify(arr,n,index);
    }
}

void buildHeap(int arr[],int n){
    for(int i=n/2;i>0;i--){
        heapify(arr,n,i);
    }
}

int main()
{
    int arr[]={-1,12,15,13,11,14};
    int n=5;
    buildHeap(arr,n);

    cout<<"print the build heap "<<endl;
    for(int i=0;i<=n;i++){
        cout<<arr[i]<<" ";
    }
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
print the build heap
-1 15 14 13 11 12
```

Tc=O(n)

Q) write program for heap sort

```cpp
#include <iostream>

using namespace std;

void heapify(int arr[],int n,int i){
    int index=i;
    int leftIndex=2*i;
    int rightIndex=2*i+1;

    int larget =index;

    if(leftIndex <=n && arr[larget] < arr[leftIndex]){
        larget=leftIndex;
    }
    if(rightIndex <=n && arr[larget] < arr[rightIndex]){
        larget=rightIndex;
    }

    if(index!=larget){
        swap(arr[index],arr[larget]);
        index=larget;
        heapify(arr,n,index);
    }
}
void buildHeap(int arr[],int n){
    for(int i=n/2;i>0;i--){
        heapify(arr,n,i);
    }
}

void heapSort(int arr[],int n){
    int index=n;
    while(n!=1){
        swap(arr[1],arr[index]);
        index--;
        n--;
        heapify(arr,n,1);
    }
}
```

```cpp
int main()
{
    int arr[]={-1,12,15,13,11,14};
    int n=5;
    buildHeap(arr,n);

    cout<<"print the build heap "<<endl;
    for(int i=0;i<=n;i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;

    heapSort(arr,n);
    cout<<"print the  heap sort "<<endl;
    for(int i=0;i<=n;i++){
        cout<<arr[i]<<" ";
    }
    return 0;
}
```

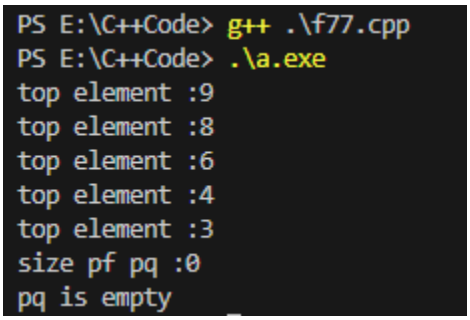
```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
print the build heap
-1 15 14 13 11 12
print the  heap sort
-1 11 12 13 14 15
```

Tc=O(nlogn)

Heaps Class -2

Q) write basic program of priority_queue(max_heap)

```cpp
#include <iostream>
#include <queue>

using namespace std;


int main()
{
    priority_queue<int>pq;
    pq.push(3);
    pq.push(6);
    pq.push(9);
    pq.push(4);
    pq.push(8);

    cout<<"top element :"<<pq.top()<<endl;
    pq.pop();

    cout<<"top element :"<<pq.top()<<endl;
    pq.pop();

    cout<<"top element :"<<pq.top()<<endl;
    pq.pop();

    cout<<"top element :"<<pq.top()<<endl;
    pq.pop();

    cout<<"top element :"<<pq.top()<<endl;
    pq.pop();

    cout<<"size pf pq :"<<pq.size()<<endl;

    if(pq.empty()){
        cout<<"pq is empty"<<endl;
    }
    else{
        cout<<"pq is NOT empty"<<endl;
    }
    return 0;
}
```


PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
top element :9
top element :8
top element :6
top element :4
top element :3
size pf pq :0
pq is empty

Q) write basic program of priority_queue (min_heap)

```cpp
#include <iostream>
#include <queue>

using namespace std;

int main()
{
    priority_queue<int,vector<int>,greater<int> >pq;
    pq.push(3);
    pq.push(6);
    pq.push(9);
    pq.push(4);
    pq.push(8);

    cout<<"top element :"<<pq.top()<<endl;
    pq.pop();

    cout<<"top element :"<<pq.top()<<endl;
    pq.pop();

    cout<<"top element :"<<pq.top()<<endl;
    pq.pop();

    cout<<"top element :"<<pq.top()<<endl;
    pq.pop();

    cout<<"top element :"<<pq.top()<<endl;
    pq.pop();

    cout<<"size pf pq :"<<pq.size()<<endl;

    if(pq.empty()){
        cout<<"pq is empty"<<endl;
    }
    else{
        cout<<"pq is NOT empty"<<endl;
    }
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
top element :3
top element :4
top element :6
top element :8
top element :9
size pf pq :0
pq is empty
```

Q) write program for kth smallest element

```cpp
#include <iostream>
#include <queue>

using namespace std;

int kthSmallestElement(int arr[],int n,int k){

    priority_queue<int>pq;

    for(int i=0;i<k;i++){
        pq.push(arr[i]);
    }

    for(int j=k;j<n;j++){

        int element=arr[j];

        if(element < pq.top()){
            pq.pop();
            pq.push(element);
        }
    }
    int ans=pq.top();
    return ans;
}
int main()
{
    int arr[]={50,2,60,7,9};
    int n=5;
    int k=3;
    int ans=kthSmallestElement(arr,n,k);
    cout<<"ans is "<<ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
ans is 9
```

Tc=O(n)

Sc=O(1)

2nd approach using min heap

```cpp
#include <iostream>
#include <queue>

using namespace std;

int kthSmallestElementUsingMinHeap(int arr[],int n,int k){

    priority_queue<int,vector<int>,greater<int> >pq;

    for(int i=0;i<k;i++){
        pq.push(arr[i]);
    }

    for(int j=k;j<n;j++){

        int element=arr[j];

        if(element > pq.top()){
            pq.pop();
            pq.push(element);
        }
    }
    int ans=pq.top();
    return ans;
}
int main()
{
    int arr[]={50,2,60,7,9};
    int n=5;
    int k=3;
    int ans=kthSmallestElementUsingMinHeap(arr,n,k);
    cout<<"ans is "<<ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
ans is 9
```

Tc=O(n)

Sc=O(n)

Heaps Class -3

Q) merge K sorted array

```
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

class info{
    public:
        int data;
        int row;
        int col;
    info(int val,int r,int c){
        data=val;
        row=r;
        col=c;
    }
};

class compare{
    public:
    bool operator()(info* a,info* b){
        return a->data > b->data;
    }
};

vector<int> mergeKSortedArray(int arr[][4],int k,int n){
    priority_queue< info* ,vector<info*>,compare > minHeap;

    for(int i=0;i<k;i++){
        info* temp=new info(arr[i][0],i,0);
        minHeap.push(temp);
    }

    vector<int>ans;

    while(!minHeap.empty()){
        info* temp=minHeap.top();
        int topElement=temp->data;
        int topRow=temp->row;
        int topCol=temp->col;
        minHeap.pop();
```

```
        ans.push_back(topElement);

        if(topCol +1 <n){
            info* newInfo=new info(arr[topRow][topCol+1],topRow,topCol+1);
            minHeap.push(newInfo);
        }
    }
    return ans;
}

int main()
{
    int arr[][4]={{2,4,6,8},
                  {1,3,5,7},
                  {0,9,10,11}

    };
    int k=3;
    int n=4;

    vector<int>ans=mergeKSortedArray(arr,k,n);
    for(auto i:ans){
        cout<<i<<" ";
    }
    cout<<endl;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
0 1 2 3 4 5 6 7 8 9 10 11
```

Tc=O(nklog)

Description | Editorial | Solutions (8.6K) | Submissions

23. Merge k Sorted Lists

Hard | 17.4K | 630

Companies

You are given an array of `k` linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

```
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6
```

C++ | Auto

```cpp
class compare{
   public :
      bool operator()(ListNode* a,ListNode* b){
          return a->val > b->val;
      }
};
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        priority_queue<ListNode*, vector<ListNode*>, compare>  minHeap;

        int k=lists.size();
        if(k==0){
            return NULL;
        }

        for(int i=0;i<k;i++){
            if(lists[i]!=NULL){
                minHeap.push(lists[i]);
            }
        }

        ListNode* head=NULL;
        ListNode* tail=NULL;

        while(!minHeap.empty()){
```

Console | Run

Description | Editorial | Solutions (753) | Submissions

632. Smallest Range Covering Elements from K Lists

Hard | 3K | 51

microsoft

Companies

You have `k` lists of sorted integers in **non-decreasing order**. Find the **smallest** range that includes at least one number from each of the `k` lists.

We define the range `[a, b]` is smaller than range `[c, d]` if `b - a < d - c` **or** `a < c` if `b - a == d - c`.

Example 1:

```
Input: nums = [[4,10,15,24,26],[0,9,12,20],[5,18,22,30]]
Output: [20,24]
Explanation:
List 1: [4, 10, 15, 24,26], 24 is in range [20,24].
List 2: [0, 9, 12, 20], 20 is in range [20,24].
List 3: [5, 18, 22, 30], 22 is in range [20,24].
```

Example 2:

C++ | Auto

```cpp
class Node{
    public :
    int data;
    int row;
    int col;

    Node(int d,int r,int c){
        data=d;
        row=r;
        col=c;
    }
};
class Compare{
    public :
    bool operator()(Node* a,Node* b){
        return a->data > b->data;
    }
};
class Solution {
public:
    vector<int> smallestRange(vector<vector<int>>& nums) {
        int mini=INT_MAX;
        int maxi=INT_MIN;

        priority_queue<Node*, vector<Node*>,Compare> minHeap;

```

Console

Heaps Class – 4

1962. Remove Stones to Minimize the Total

Hint

Medium 1.7K 138

Companies

You are given a **0-indexed** integer array `piles`, where `piles[i]` represents the number of stones in the i^{th} pile, and an integer `k`. You should apply the following operation **exactly** `k` times:

- Choose any `piles[i]` and **remove** `floor(piles[i] / 2)` stones from it.

Notice that you can apply the operation on the **same** pile more than once.

Return *the **minimum** possible total number of stones remaining after applying the* `k` *operations.*

`floor(x)` is the **greatest** integer that is **smaller** than or **equal** to `x` (i.e., rounds `x` down).

T(:O(n+Klog(n))

Example 1:

Input: piles = [5,4,9], k = 2

```cpp
class Solution {
public:
    int minStoneSum(vector<int>& piles, int k) {
        priority_queue<int> maxHeap;

        for(int i=0;i<piles.size();i++){
            maxHeap.push(piles[i]);
        }

        while(k--){
            int maxElement=maxHeap.top();
            maxHeap.pop();
            maxElement=maxElement-floor(maxElement/2);
            maxHeap.push(maxElement);
        }

        int sum=0;
        while(!maxHeap.empty()){
            int temp=maxHeap.top();
            maxHeap.pop();
            sum+=temp;
        }
        return sum;
    }
};
```