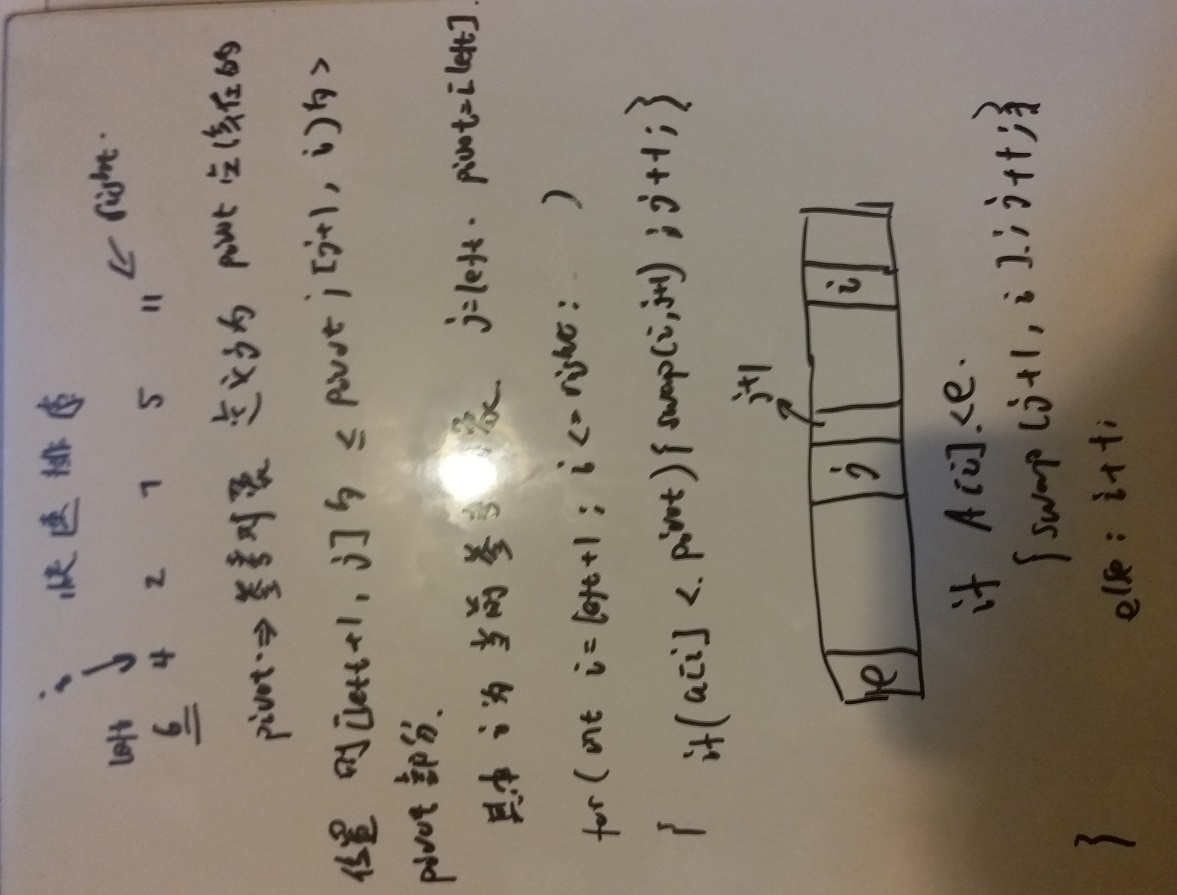
**排序算法总结:**

**快速排序思路以及图示:**



代码如下:

测试部分代码:

**package** algorithm;

**public** **class** Main {

**public** **static** **void** main(String[] args)

{

**int**[] nums={3,2,4,5,5,6,7,1,0,-2};

**new** quicksort().qsort(nums, 0, nums.length-1);

**for**(**int** e:nums)

{

System.***out***.print(e+" ");

}

}

}

算法部分代码:

**package** algorithm;

**public** **class** quicksort {

**void** qsort(**int**[] nums,**int** left,**int** right)

{

**if**(left>=right)**return**;

**int** j=left;

**int** pivot=nums[j];

/\*

\* 定义j为分界点，i为当前考察元素

\* [left+1,j]的范围内的值都小于等于pivot

\* [j+1,i)范围内的部分都>pivot

\*/

**for**(**int** i=left+1;i<=right;i++)

{

**if**(nums[i]<=pivot)

{

**int** temp=nums[j+1];

nums[j+1]=nums[i];

nums[i]=temp;

j++;

}

}

/\*

\* swap j and pivot;

\*/

nums[left]=nums[j];

nums[j]=pivot;

/\*

\* recursively do that

\*/

qsort(nums,left,j-1);

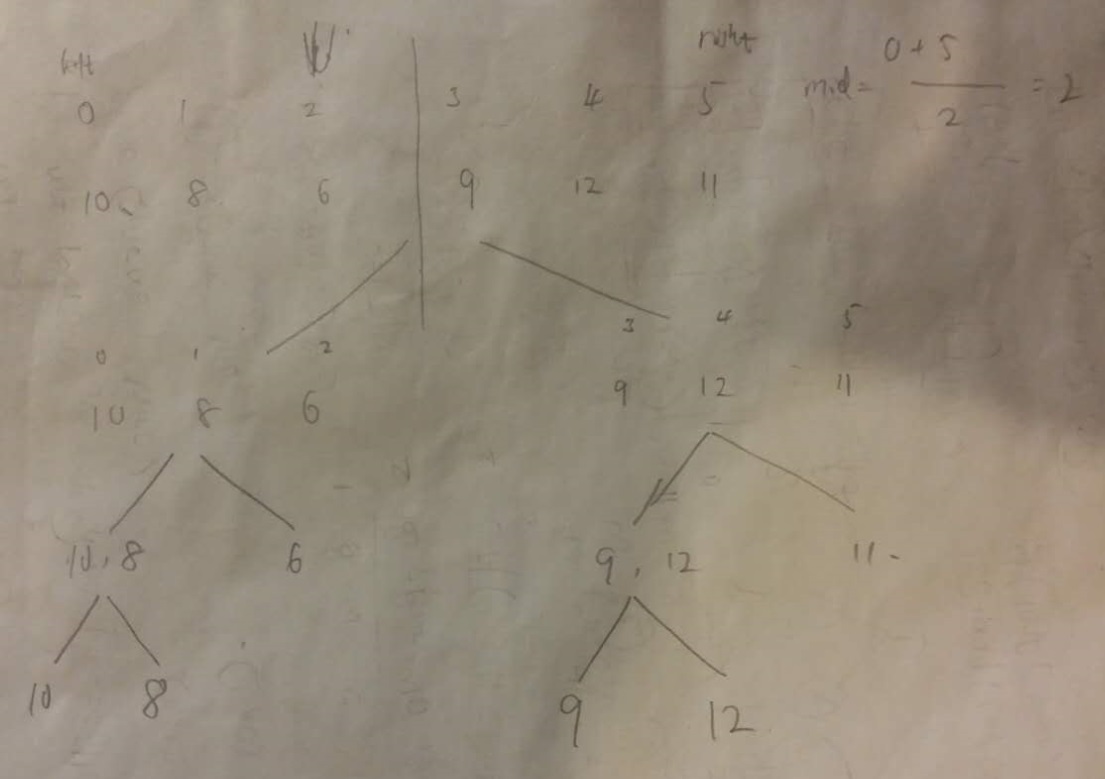
qsort(nums,j+1,right);

}

}

**Merge Sort归并排序的思路：**

**其实就是分治的思想，将数组一直往下分，分到只剩下1个元素或者两个元素的时候，可以进行比较排序，然后再往上进行合并，divide,conquer,最后的combine是难点，需要借助辅助数组实现**

****

**代码部分：**

**package** algorithm;

**public** **class** MergeSort {

**public** **void** mergeSort(**int** left,**int** right,**int**[] nums)

{

**if**(left>=right)**return**;

**else**

{

**int** middle=(right+left)/2;

mergeSort(left,middle,nums);

mergeSort(middle+1,right,nums);

merge(left,middle,right,nums);

}

}

**public** **void** merge(**int** left,**int** mid,**int** right,**int**[] nums)

{

**int**[] aux=**new** **int**[right-left+1];

**for**(**int** i=left;i<=right;i++)

{

aux[i-left]=nums[i];

}

**int** i=left;

**int** j=mid+1;

**for**(**int** k=left;k<=right;k++)

{

**if**(i>mid)

{

nums[k]=aux[j-left];

j++;

}

**else** **if**(j>right)

{

nums[k]=aux[i-left];

i++;

}

**else**

{

**if**(aux[i-left]<aux[j-left])

{

nums[k]=aux[i-left];

i++;

}

**else**

{

nums[k]=aux[j-left];

j++;

}

}

}

}

}

**测试部分：**

**package** algorithm;

**public** **class** Main {

**public** **static** **void** main(String[] args)

{

**int**[] nums={4,3,6,2,1,9,8,10};

**new** MergeSort().mergeSort(0, nums.length-1, nums);

**for**(Integer e:nums)

System.***out***.print(e+" ");

}

}

**插入排序思路和实现：**

其本质还是交换排序的思路和方法

**package** algorithm;

**public** **class** InsertionSort {

**public** **void** insertionSort(**int**[] arr)

{

**if**(arr==**null**||arr.length<=1)**return**;

**else**

{

**for**(**int** i=1;i<arr.length;i++)

{

**for**(**int** j=i;j>=1;j--)

{

**if**(arr[j]<=arr[j-1])

{

**int** temp=arr[j];

arr[j]=arr[j-1];

arr[j-1]=temp;

}

**else**

{

**break**;

}

}

}

**return**;

}

}

}

测试部分代码：

**package** algorithm;

**public** **class** Main {

**public** **static** **void** main(String[] args)

{

**int**[] nums={4,3,6,2,1,9,8,10};

**new** InsertionSort().insertionSort(nums);;

**for**(Integer e:nums)

System.***out***.print(e+" ");

}

}

改进效率的方法：把交换操作换成赋值操作：

**package** algorithm;

**public** **class** InsertionSort {

**public** **void** insertionSort(**int**[] arr)

{

**if**(arr==**null**||arr.length<=1)**return**;

**else**

{

**for**(**int** i=1;i<arr.length;i++)

{

**int** j=i;

**int** e=arr[i];

**for**(j=i;j>=1;j--)

{

**if**(arr[j-1]>=e)

{

arr[j]=arr[j-1];

}

**else** **break**;

}

arr[j]=e;

}

**return**;

}

}

}

**冒泡排序算法：**

**第一次最大的到最后，第二次次大，直到只有两个元素为止**

**代码：**

**package** algorithm;

**public** **class** BubbleSort {

**public** **void** bubbleSort(**int**[] arr)

{

**if**(arr==**null**||arr.length<=1)**return**;

**for**(**int** j=arr.length-1;j>=1;j--)

**for**(**int** i=0;i<=arr.length-2;i++)

{

**if**(arr[i]>arr[i+1])

{

**int** temp=arr[i];

arr[i]=arr[i+1];

arr[i+1]=temp;

}

}

**return**;

}

}

**测试代码：**

**package** algorithm;

**public** **class** Main {

**public** **static** **void** main(String[] args)

{

**int**[] nums={4,3,6,2,1,9,8,10};

**new** BubbleSort().bubbleSort(nums);

**for**(Integer e:nums)

System.***out***.print(e+" ");

}

}

**九章算法课程总结:**

二分搜索和排序数组问题:

经典二分搜索的模型:给定一个数组，已经排好序，和一个target,然后找到target在数组中出现的第一个，最后一个位置等，不存在的时候返回-1.

例题1: 搜索第一次出现的位置:

0,1,2,3,4,5,6

[1,2,3,3,4,5,10]

**Easy 难度的题目，抽取模板:**

class Solution {

/\*\*

\* @param nums: The integer array.

\* @param target: Target to find.

\* @return: The first position of target. Position starts from 0.

\*/

public int binarySearch(int[] nums, int target) {

//write your code here

//it includes the duplicate element in the array

int start = 0;

int end = nums.length - 1;

while(start < end){

int mid = start + (end - start)/2;

if(nums[mid] == target){

end = mid;

}

else if(nums[mid] < target){

start = mid + 1;

}

else {

end = mid - 1;

}

}

if(nums[start] == target)return start;

else return -1;

}

}

**二分法模板:**

对于你去查找某一个数的时候，start和end确实是可以使用+1,-1的方式，反之如果不是的话，那么这个时候不好判定，所以模板方法1 就是把终止条件改成

While(start + 1 < end),也就是说当最后只有两个数的时候进行退出

**关键点:**

1.Start + 1 < end

2.Mid = start + (end - start) /2;

3.> < ==

4.target == [start] ? target == [end] ?

**Search in range 问题:**

1. find the first position 2. Find the right position

使用两次二分搜索分别找到这两个位置，然后就完成了，所以代码可以这样写:

**public class Solution {**

**/\*\***

**\*@param A : an integer sorted array**

**\*@param target : an integer to be inserted**

**\*return : a list of length 2, [index1, index2]**

**\*/**

**public int[] searchRange(int[] A, int target) {**

**// write your code here**

**if(A == null || A.length ==0){**

**return new int[]{-1,-1};**

**}**

**int start = 0,end = A.length - 1;**

**while(start + 1 < end){**

**int mid = start + (end - start)/2;**

**if(A[mid] == target){**

**end = mid;**

**}**

**else if(A[mid] > target){**

**end = mid;**

**}**

**else{**

**start = mid;**

**}**

**}**

**int[] result = new int[2];**

**if(A[start] == target){**

**result[0] = start;**

**}**

**else if(A[end] == target){**

**result[0] = end;**

**}**

**else{**

**return new int[]{-1,-1};**

**}**

**start = 0;end = A.length - 1;**

**while(start + 1 < end){**

**int mid = start + (end - start)/2;**

**if(A[mid] == target){**

**start = mid;**

**}**

**else if(A[mid] > target){**

**end = mid;**

**}**

**else{**

**start = mid;**

**}**

**}**

**if(A[end] == target){**

**result[1] = end;**

**}**

**else if (A[start] == target){**

**result[1] = start;**

**}**

**return result;**

**}**

**}**

自己写的出错代码:

public class Solution {

/\*\*

\*@param A : an integer sorted array

\*@param target : an integer to be inserted

\*return : a list of length 2, [index1, index2]

\*/

public int[] searchRange(int[] A, int target) {

// write your code here

if(A == null || A.length ==0){

return new int[]{-1,-1};

}

int start = 0;

int end = A.length - 1;

while(start + 1 < end){

int mid = start + (end - start)/2;

if(A[mid] < target){

start = mid;

}

else if(A[mid] > target){

end = mid;

}

else{

end = mid;

}

}

int[] result = new int[2];

if(A[start] == target){

result[0] = start;

}

else if(A[end] == target){

result[0] = end;

}

else{

return new int[]{-1,-1};

}

start = start + 1;

end = A.length - 1;

while(start + 1 < end){

int mid = start + (end - start)/2;

if(A[mid] == target){

start = mid;

}

else if(A[mid] > target){

start = mid;

}

else{

end = mid;

}

}

if(A[end] == target){

result[1] = end;

return result;

}

else if(A[start] == target){

result[1] = start;

return result;

}

return new int[]{-1,-1};

}

}

按照上边那个模板进行总结就是:

如果你想找第一次出现的，那么找到目标时候就让你end = mid; 同时在循环结束时候先判断start的值

如果你想找最后一次出现的，那么同样的道理就是让start = mid 同时在循环结束时候先判断end的值

上边这个题目实际上就是综合了这两种情况，因此两次二分搜索，找到那个界.

而上边问题的共同点都是在找等于给定值的情况。

现在问题实际上可以有一些改变，当找的东西不再是一个特定的数字的时候，应该怎么处理:

**例如:Search Insert Position:**

public class Solution {

/\*\*

\* param A : an integer sorted array

\* param target : an integer to be inserted

\* return : an integer

\*/

public int searchInsert(int[] A, int target) {

// write your code here

if(A == null || A.length == 0)return 0;

int start = 0;

int end = A.length - 1;

while(start + 1 < end){

int mid = start + (end - start)/2;

if(A[mid] <= target){

start = mid;

}

else{

end = mid;

}

}

if(target > A[start] && target < A[end]) return start+1;

else if(target == A[start]) return start;

else if(target < A[start]) return start;

else if(target == A[end]) return end;

else return end + 1;

}

}

关键在于如何描述 insert position 这个定义：

1. 找到最后一个比target 小或者等于的位置
2. 找到第一个比target 大或者等于的位置

而我上边的解法实际上就是在找最后一个比target小于等于的位置

下边我再写一个第一个比target 大于等于的位置的方法:

public class Solution {

/\*\*

\* param A : an integer sorted array

\* param target : an integer to be inserted

\* return : an integer

\*/

public int searchInsert(int[] A, int target) {

// write your code here

if(A == null || A.length == 0 )return 0;

int start = 0;

int end = A.length - 1;

while(start + 1 < end){

int mid = start + (end - start) / 2;

if(A[mid] >= target){

end = mid;

}

else{

start = mid;

}

}

//下边这部分对应的代码其实和之前的一样

if(A[start] == target) return start;

else if(A[start] > target)return start;

else if(target > A[start] && target < A[end])return start + 1;

else if(target == A[end]) return end;

else return end + 1;

}

}

规律就是说start和end中二者必有一个满足是第一个>= target的值.

**练习:search 2d matrix 和search 2d matrix II**

**Binary Tree , Divide and conquer, DFS ,BFS**

Design pattern 问题:singleton,factory,what is design pattern

Data base 问题:classification of data base type,left join,right join,search …

Multi thread,operation system 问题

**Binary Tree DFS traverse :**

**Pre order/in order/post order**

**Divide and conquer**

**DFS template**

**Binary tree bfs traverse**

**Bfs template**

**Binary search tree**

**Validate insert,delete**

**Divide and conquer algorithm:**

**Merge sort**

**Quick sort**

**Most of the binary tree program**

**例题:关于归并排序的题目:耗费额外空间o（n）特点：先局部有序然后总体有序**

**Quick sort:先整体有序然后局部有序**

**区别在于进行排序时候的稳定性上边merge sort 稳定排序而quick sort 则属于不稳定的排序方法**

**例题:二叉树的最大结点:**

public class Solution {

/\*\*

\* @param root the root of binary tree

\* @return the max ndoe

\*/

public TreeNode maxNode(TreeNode root) {

// Write your code here

if(root==null)return null;

TreeNode left=maxNode(root.left);

TreeNode right=maxNode(root.right);

TreeNode temp=null;

if(left!=null&&right!=null)

{

temp=(left.val>right.val?left:right);

if(temp.val<root.val){

temp=root;

}

}

else if(left==null && right!=null){

temp=(right.val>root.val?right:root);

}

else if(left!=null && right==null){

temp=(left.val>root.val?left:root);

}

else{

temp=root;

}

return temp;

}

}

**分治的方法求解二叉树的最大深度:**

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: The root of binary tree.

\* @return: An integer.

\*/

public int maxDepth(TreeNode root) {

// write your code here

if(root==null)return 0;

int left=maxDepth(root.left);

int right=maxDepth(root.right);

return 1+Math.max(left,right);

}

}

**另外一个分治的典型例子：平衡二叉树:**

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: The root of binary tree.

\* @return: True if this Binary tree is Balanced, or false.

\*/

public boolean isBalanced(TreeNode root) {

// write your code here

if(root==null)return true;

int ldepth=depthFromRoot(root.left);

int rdepth=depthFromRoot(root.right);

if(Math.abs(ldepth-rdepth)>1)return false;

return isBalanced(root.left) && isBalanced(root.right);

}

private int depthFromRoot(TreeNode root){

if(root==null)return 0;

int left=depthFromRoot(root.left);

int right=depthFromRoot(root.right);

return 1+Math.max(left,right);

}

}

**二叉树中最大路径和:**

实际上这个是Leetcode上边的124 hard 难度的题目

对于根结点root来说，最大路径有三种:

1. 经过root从左边结点到右边结点
2. 不经过root，完全在root的左半部分
3. 不经过root,完全在root的右半部分

返回的结果就是max(1,2,3)这三种情况的最大值:

所以需要定义一个函数int MaxLenStartFrom(TreeNode root)

这个函数表示的含义就是从root出发的到其某一个结点的最大值

错误代码:

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: The root of binary tree.

\* @return: An integer.

\*/

public int maxPathSum(TreeNode root) {

// write your code here

if(root==null)return 0;

if(root.left==null && root.right==null)return root.val;

if(root.left!=null && root.right==null){

int leftValue=maxPathSum(root.left);

return leftValue+root.val;

}

else if(root.left==null && root.right!=null){

int rightValue=maxPathSum(root.right);

return rightValue+root.val;

}

int leftValue=maxPathSum(root.left);

int rightValue=maxPathSum(root.right);

int mid=MaxValueStartFrom(root.left)+MaxValueStartFrom(root.right)+root.val;

return Math.max(leftValue,Math.max(mid,rightValue));

}

private int MaxValueStartFrom(TreeNode root){

if(root==null)return Integer.MIN\_VALUE;

int left=MaxValueStartFrom(root.left);

int right=MaxValueStartFrom(root.right);

if(left==Integer.MIN\_VALUE && right == Integer.MIN\_VALUE)return root.val;

return root.val+Math.max(left,right);

}

}

**代码具体如下:**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

class NodeType{

int downValue;

int maxValue;

public NodeType(){

}

public NodeType(int down,int max){

this.downValue=down;

this.maxValue=max;

}

}

public NodeType getMax(TreeNode root){

if(root==null){

return new NodeType(0,Integer.MIN\_VALUE);

}

//divide it into two part

NodeType left=getMax(root.left);

NodeType right=getMax(root.right);

//then we see for the current node info

NodeType cur = new NodeType();

//for the single path,if the down path is negative ,we don't want it ,then set it to zero,or it is positive or zero,we can take that

cur.downValue=root.val+Math.max(left.downValue,right.downValue);

cur.downValue=Math.max(cur.downValue,0);

//for the max value,the topic say that there must be one node here,so it can come from three:

//1.pure left max

//2.pure right max

//3.the one include single left+root+single right

//we get the maximum one for that

cur.maxValue=Math.max(left.maxValue,right.maxValue);

cur.maxValue=Math.max(root.val+left.downValue+right.downValue,cur.maxValue);

return cur;

}

public int maxPathSum(TreeNode root) {

NodeType result=getMax(root);

return result.maxValue;

}

}

**最近公共祖先:**

**无parent的情况:**

**函数的定义:如果以root为根的子树中同时存在Node1，node2返回其公共祖先,如果root==node1,node2任意一个，返回root.如果不含有node1,node2,那么返回空**

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: The root of the binary search tree.

\* @param A and B: two nodes in a Binary.

\* @return: Return the least common ancestor(LCA) of the two nodes.

\*/

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode A, TreeNode B) {

// write your code here

if(root==null)return null;

if(root==A||root==B)return root;

TreeNode left=lowestCommonAncestor(root.left,A,B);

TreeNode right=lowestCommonAncestor(root.right,A,B);

if(left!=null && right!=null)return root;

else if(left==null && right !=null)return right;

else if(left!=null && right==null) return left;

else return null;

}

}

**二叉树的宽度优先搜索:**

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: The root of binary tree.

\* @return: Level order a list of lists of integer

\*/

public ArrayList<ArrayList<Integer>> levelOrder(TreeNode root) {

// write your code here

ArrayList<ArrayList<Integer>> res=new ArrayList<ArrayList<Integer>>();

if(root==null)return res;

Queue<TreeNode> queue=new LinkedList<TreeNode>();

queue.offer(root);

while(queue.isEmpty()==false){

int size=queue.size();

List<Integer> temp=new ArrayList<Integer>();

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

TreeNode tmpNode=queue.peek();

if(tmpNode.left!=null){

queue.offer(tmpNode.left);

}

if(tmpNode.right!=null){

queue.offer(tmpNode.right);

}

temp.add(queue.poll().val);

}

res.add(new ArrayList<Integer>(temp));

}

return res;

}

}

**二叉查找树:**