链表

19. Remove Nth Node From End of List **删除倒数第 K 个节点。快先走 k,然后一起走，删除第 K 个**

21. Merge Two Sorted Lists **合并两个有序链表，可以递归，可以非递归。**l1.next = mergeTwoLists(l1.next,l2);

```
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if(l1 == null) return l2;
        if(l2 == null) return l1;
        if(l1.val < l2.val){
            l1.next = mergeTwoLists(l1.next,l2);
            return l1;
        }else{
            l2.next = mergeTwoLists(l1,l2.next);
            return l2;
        }
    }
```

24. Swap Nodes in Pairs **成对翻转链表：**pre.next!=null &&pre.next.next!=null 时，将 cur.next 插入到 pre 之后。

61. Rotate List **链表右移 K 位**:遍历求 len,从 dummy 走 len-k 步，后面的直接放 dummy 后，尾指头，真尾空

82. Remove Duplicates from Sorted List II:**将重复的全部删掉。**

83. Remove Duplicates from Sorted List，**将重复多余的删掉，保留一个。**

```
while (cur!=null){
    if(cur.val == pre.val){
        pre.next = cur.next;
    }else{
        pre = pre.next;
    }
    cur = cur.next；}
```

86. Partition List：将链表比 target 小的左移放前面，后面的都是大于等于的，两边相对位置不变。

建立两个头结点，遍历原链表，小的放在第一后面，大的放在第二个后面。然后将大的放入小后。

92.Reverse Linked List II,**将 m-n 之间的链表逆转。在 m-n 之间进行头插法。**

先走 m 步，然后头插法反转 m-n 次。

109. Convert Sorted List to Binary Search Tree：**链表中间节点作为根节点，前面的左子树，右边的右子树。一快一慢指针，走到中间。生成根节点，左边的断开，生成左子树，右边的生成右子树**

```
public TreeNode helper109(ListNode head){
 if(head == null)
     return null;
 ListNode dummy = new ListNode(0);
 dummy.next = head;
 ListNode pre = dummy;
 ListNode slow = head;
 ListNode fast = head;
 while(fast.next!=null && fast.next.next!=null){
     fast =fast.next.next;
     slow = slow.next;
     pre = pre.next;
 }
```

```
        TreeNode root= new TreeNode(slow.val);
        pre.next = null;
        root.left =helper109(dummy.next);
        root.right = helper109(slow.next);
        return root;
    }
```

138. Copy List with Random Pointer:链表有 next,有 random;复制他， 用 hashmap;

复杂链表复制，用 hashMap<Node,new Node(val)>;然后遍历原链表；get(node).next =

141. Linked List Cycle:链表是否有环，fast 走两步，slow 走一步。如果相遇或者到 null,出结果。

142. Linked List Cycle II，链表入口节点位置。Fast，slow 相遇，然后 fast 从 head 出发，再相遇。

143. Reorder List：第一个-倒 1-第 2-倒 2；这个比较麻烦！

快慢指针，找到中间的节点。然后将中间之后的节点采用头插法反转链表。然后前面的和后面的进行遍历，将后面的一个个的插入前面的中间。

```
        while (cur2!=null){
            ListNode next1 = cur1.next;
            ListNode next2 = cur2.next;
            cur1.next = cur2;
            cur2.next = next1;
            cur1 = next1;
            cur2 = next2;
        }
```

147. Insertion Sort List:插入排序重排链表

插入排序，while(cur.next!=null){将 cur.next 的值和 cur 比较，如果比他大不用排，继续。否则 p 从 dummy 遍历，找到 p.next>next.val 的，将 next 插入 p 后面。整个就是插入排序的思路。}

148. Sort List，NlogN 给链表排序：分治：将链表切成两部分，然后递归求这个。然后 merge;

归并算法：首先检查是否链表长度为 null 或者为 1；如果是直接返回，不是就切割成两半，用快慢指针快慢指针。不停的且一半有一半切割 l1 = sortList(head); l2 = sortList(solw);然后并，并的过程就是有序链表合并;

160. Intersection of Two Linked Lists,第一个长度，第二个长度，然后走齐，然后一起走到一样.

是否有共同点，看最后是否一致。

找共同点，l1 遍历，空继续 l2. l2 遍历，空找 l1.直到两人一样。

234：链表是否是回文：

快慢找到中间，然后逆转后面的，前后齐步走，看看是否始终值一样

328. Odd Even Linked List：偶数位的位于前面，奇数位的后面。一个 odd 开头，一个 even 开头，分别找 next.next；

```
        ListNode odd = head,even = head.next, last = even;
        while (even!=null && even.next!=null){
            odd.next = odd.next.next;
            even.next =even.next.next;
            odd = odd.next;
            even = even.next;
        }
        odd.next = last;
        return head;
```

二叉树：

94. Binary Tree Inorder Traversal：**中序遍历，非递归。一路左，弹出，cur=pop.right;**

```
public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> resList = new ArrayList<>();
        if(root == null) return resList;
        Stack<TreeNode> stack = new Stack<>();
        while(root != null || !stack.isEmpty()){
            while(root!=null){
                stack.push(root);
                root = root.left;
            }
            root = stack.pop();
            resList.add(root.val);
            root = root.right;
        }
        return resList;
    }
```

96. Unique Binary Search Trees:**N 个节点，有几种 BST,动态规划规律可解**

98. Unique Binary Search Trees：**是不是 BST,中序遍历是递增的，保持前序的节点；然后比较大小**

```
中序遍历，维护一个 pre 表示上一个节点，比较当前和之前，如实当前不大于之前，不是 BST;
if(pre == null) pre = cur;
else if(pre.val >= cur.val) return false;
pre = cur;
```

100. Same Tree：**俩树是否一致。递归。都空返回 true,一个空或者值不一样 false;然后比较左子树和右**

101. Symmetric Tree：**是否对称，和上个一样，转化成递归(root.left,root.right);**

```
Symmetric(root1.left,root2.right) && Symmetric(root1.right,root2.left);
```

102. Binary Tree Level Order Traversal:**分层，分开打印。Size**

103. Binary Tree Zigzag Level Order Traversal 之字形，**size+flag**

```
if(!flag) Collections.reverse(tempList);
resList.add(new ArrayList(tempList));
flag = !flag;
```

104. Maximum Depth of Binary Tree:**递归，或者层序 size+level**

105. Construct Binary Tree from Preorder and Inorder Traversal:中序放入 map(in[i],i);

106. Construct Binary Tree from Inorder and Postorder Traversal:中序放入 map(in[i],i);

107. Binary Tree Level Order Traversal II:**从下层到顶层的层序；size + res.add(0,levelList);**

108. Convert Sorted Array to Binary Search Tree：**有序数组转化成二叉树，中间元素作根节点，其他的递归**

```
public TreeNode helper(int[]nums,int left,int right){
        if(left > right)    return null;
        int mid = left + (right -left)/2;
        TreeNode root = new TreeNode(nums[mid]);
        root.left = helper(nums,left,mid-1);
        root.right = helper(nums,mid+1,right);
        return root;
    }
```

110. Balanced Binary Tree：求深度的时候，作 left 和 right 的差值比较；

111. Minimum Depth of Binary Tree：**最小深度，层序，size+遇到叶节点返回 level；**

112. Path Sum：根节点到叶节点是否之和是否存在等于 target 的，直接 hasPathSum，递归；

```
public boolean hasPathSum(TreeNode root, int sum) {
        if(root == null)    return false;
        if(root.val == sum && root.left == null && root.right == null) return true;
        return hasPathSum(root.left,sum-root.val) || hasPathSum(root.right,sum-root.val);
    }
```

113. Path Sum II：给出所有的根到叶之和为 target 的组合。helper113(root,sum,resList,temList);

```
if(root == null) return;
tempList.add(root.val);
if(root.left == null && root.right == null && sum == root.val)
    resList.add(new ArrayList(tempList));
helper(resList,tempList,root.left,sum-root.val);
helper(resList,tempList,root.right,sum-root.val);
tempList.remove(tempList.size()-1);
```

114. Flatten Binary Tree to Linked List：所有节点飘向右方

遍历根节点，如果有左子树，那么将 root.right = root.left;root.left =null;就是将整个左子树移到右子树上。但原本的右子树和根的左子树如何连一块？将右子树放到左子树最右边的那个节点的右边。

```
while(root != null){
    if(root.left != null){
        TreeNode prev = root.left;
        while(prev.right != null){
            prev = prev.right;
        }
        prev.right = root.right;
        root.right = root.left;
        root.left = null;
    }
    root = root.right;
}
```

116. Populating Next Right Pointers in Each Node：完全二叉树，整个右指针指向层序的右边那个。

117. Populating Next Right Pointers in Each Node II：普通二叉树，右指针指向右边那个。

129. Sum Root to Leaf Numbers：根到叶拼接成数字，所有的加一块之和。递归 sumRootToLeaf(root,0);

```
if(root == null )    return 0;
if(root.left == null && root.right == null)
    return s*10+root.val;
return sumRootToLeaf(root.left,s*10+root.val) + sumRootToLeaf(root.right,s*10+root.val);
```

144. Binary Tree Preorder Traversal：前序遍历

145，后序遍历；

```
public List<Integer> postorderTraversal(TreeNode root) {
    List<Integer> results = new ArrayList<Integer>();
    Deque<TreeNode> stack = new ArrayDeque<TreeNode>();
    while (!stack.isEmpty() || root != null) {
```

```
            if (root != null) {
                stack.push(root);
                results.add(root.val);
                root = root.right;
            } else {
                root = stack.pop().left;
            }
        }
        Collections.reverse(results);
        return results;
}
```

199. Binary Tree Right Side View：层序的最后一个的节点列表。Size+level==0;

```
Queue<TreeNode> que = new LinkedList();
que.add(root);
while(!que.isEmpty()){
    int size = que.size();
    while(size>0){
        TreeNode node = que.poll();
        if(size==1)
            result.add(node.val);
        if(node.left != null)
            que.add(node.left);
        if(node.right != null)
            que.add(node.right);
        size--;
    }
```

222. Count Complete Tree Nodes:完全二叉树的节点个数；

因为是完全二叉树，所以可以查看最左节点深度和最右节点深度，如果一样，说明是满了，2^n-1;

否则，1+递归左子树+递归右子树。时间复杂度？lgn ^ 2;

```
public int countNodes(TreeNode root) {
    int leftDepth = leftDepth(root);
    int rightDepth = rightDepth(root);
    if(leftDepth == rightDepth)
        return (1<<leftDepth) -1;
    return 1+countNodes(root.right)+countNodes(root.left);
}
public int leftDepth(TreeNode root){
    int depth =0;
    while (root!=null){
        depth++;
        root = root.left;
    }
    return depth;
}
```

```
public int rightDepth(TreeNode root){
    int depth =0;
    while (root!=null){
        depth++;
        root = root.right;
    }
    return depth;
}
```

230. Kth Smallest Element in a BST：BST 第 K 小的节点，**中序遍历，**

235. Lowest Common Ancestor of a Binary Search Tree

二叉查找树的特点，根大于所有的左子树，根小于所有的右子树。所以只有根节点值和 p,q 比较就行。

如果 pq 都小于根，说明最近祖先在左子树。如果都大于根，说明最近祖先在右子树。否则根就是。

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    if(root.val > p.val && root.val > q.val)
        return lowestCommonAncestor(root.left,p,q);
    if(root.val < p.val && root.val < q.val)
        return lowestCommonAncestor(root.right,p,q);
    return root;
}
```

236. Lowest Common Ancestor of a Binary Tree：

从头到下递归，递归返回条件时：如果 root 遇到 p 或者 q，返回。或者为 null;如果 left 为空，说明都在右边，right 就是，如果 right 为空，那就在 left.如果都不空，说明俩边都有，返回 root;

```
if(root == null || root == p || root == q) return root;
TreeNode left = lowestCommonAncestor(root.left,p,q);
TreeNode right = lowestCommonAncestor(root.right,p,q);
return left == null ? right : right == null ? left : root;
```

543. Diameter of Binary Tree,两个节点的最长路径长度

```
public int getDepth(TreeNode root){
    if(root == null) return 0;
    int left = getDepth(root.left);
    int right = getDepth(root.right);
    max = Math.max(max,left+right);
    return Math.max(left,right)+1;
}
```

剑指 offer 里面的相关题目：

No3.数组中重复的数字 n 个数，范围 0 到 n-1,找到任意一个重复的即可

将值为 i 的数，放入 i 的位置。从位置遍历，将 A[i]和 A[A[i]]的互相交换，直到 A[i] = I;

重复的时候

空间复杂度为 N 的好办，boolean 数组即可。如果没有额外空间，就交换。

No4,二维数组查找，从左到右递增，从上到下递增。

右上开始，比 target 小，向下。比他大，就像左。

如果层铺开，有序。可以二分查找。算 mid 在二维数组中的位置

No5,空格替换，如果有空格，就将他替换成%20

(StringBuffer str)遍历得到长度 2*空格+原来的长度，str.setLength(newLength);

str.setCharAt(j--, '0');替换原来的。更新

//str.insert(0,str)插入。

No6 从尾到头打印链表

入栈，出栈；

No7 重建二叉树,前序遍历和中序遍历重构二叉树

首先将中序遍历的结果存到 map 中，map.put(in[i],i);key 为值，value 为位置。

前序第一个是根节点。然后找到中序数组中根节点的位置，然后算出左节点个数，右节点个数，递归。

!!!!! No8 二叉树的下一个节点，中序遍历的二叉树下一个节点，这个树的节点，有指向父指针;

如果二叉树该节点的右子树不为空，那么右子树的最左节点就是下一个节点。

如果为空，那么下一个节点就是右边的父节点。

```
TreeLinkNode parent = pNode.next;
 while (parent!=null && parent.left!=pNode){
      pNode = parent;
      parent = parent.next;
 }
 return parent;
```

No9,两个栈实现队列

一个入，一个出。出的不够入的入，然后出。

No10 斐波那契数列;

f1 =1,f2 = 1, fn = fn-1 + fn-2;

No 10-1,青蛙可以挑一阶，也可以 2 阶，跳到 n 阶，几种做法

No10-2,变态跳，可以跳 1-n 阶，随便跳。求 n 阶几种方法;

return 1 << (target - 1);

No11,旋转数组最小的数字：非减排序的数组，可能存在重复 // if nums[mid] > nums[right] left = mid + 1; else if(nums[mid] < nums[right]) right = mid; else right--; return left;

No12,矩阵中的路径，矩阵有字符，可以从任意开始，上下左右任意走，但不能走走过的格子，求是否存在一条包含给定字符串的路径

矩阵，双层遍历：对每个位置进行 dfs 搜索。dfs 时，维护一个矩阵，表示是否已经走过。先筛选，如 ij 边界问题和当前遍历字母和 target 路径是否一致。成功的标志是，相同的已经到头，k= nums.length-1; 成功之后，这个遍历过。然后 dfs 向四周扩散。有一个返回 true,那就是 true.然后没遍历过,返回 fasle.

No13,机器人的运动范围，从(0,0)开始，上下左右移动，但不能进入行坐标和列坐标数位之和大于 k 的格子，给定 k,能打几个格子。

从 0，0 开始 dfs.dfs 时，维护一个矩阵，看看是否曾遍历过。判断是否越界，是否遍历过，是否行列之和小于等于 k.如果是，返回 0。然后遍历过，然后四处 dfs+1;

```
if (sRow < 0 || Scol < 0 || sRow >= rows || Scol >= cols || flag[sRow][Scol] || count(sRow,Scol) > k)
     return 0;
flag[sRow][Scol] =true;
return   helper13(sRow-1,Scol,rows,cols,flag,k)//甚至可以删了后退的两个。
          +helper13(sRow+1,Scol,rows,cols,flag,k)
          +helper13(sRow,Scol-1,rows,cols,flag,k)
```

```
                +helper13(sRow,Scol+1,rows,cols,flag,k)+1;
```

No14 剪绳子，长为 n，剪 m 段（m,n 都是整数，且 m>1），所有的长度的乘积最大为多少。

```
public int cutRope(int n) {
    if(n == 2 || n == 3)   return n-1;
    int[] dp = new int[n+1];
    dp[1] = 1;dp[2] = 2;dp[3] = 3;
    for (int i = 4; i<=n;i++){
        int max = 0;
        for (int j =1;j <= i /2 ;j++){
            max = Math.max(dp[j] * dp[i-j],max);
        }
        dp[i] = max;
    }
    return dp[n];
}
```

No15,二进制中 1 的个数;

```
while (n != 0) {
    n = n & (n - 1); //  让二进制的 n 最右边的 1 变成 0；
    count++;
}
```

No16, 数值的整数次方

```
public double Power(double base, int exponent) {
    //如果指数为负，为 0，为正。
    //如果 double 的基数为 0
    //正常情况下，不能对 base 乘以 n 次。而是折半。exponent 分奇偶，也分正负。
    if (exponent == 0 && base != 0)
        return 1;
    if (exponent == 1)
        return base;
    if(base ==0 && exponent > 0)
        return 0;
    if(base == 0 && exponent <= 0)
        throw   new RuntimeException();
    double result = 1;
    int n = exponent;
    if (exponent < 0 )
        n = -exponent;
    result   = Power(base,n >> 1);
    result = result * result;
    if ((n & 1) == 1)
        result = result * base;
    if (exponent < 0)
        result = 1 /result;
```

```
        return result;
    }
```

数组：

No18-1,删除节点。给定一链表头结点和指定节点，O(1)时间内，删除指定节点

No18-2.删除重复的节点。存在的重复节点，全都删了。Cur 走到重复节点的最后一个，看是否是 pre.next;

```
while (cur.next != null && cur.val == cur.next.val){
    cur= cur.next;
}
```

No19,实现函数来匹配包含‘.’和‘*’的正则表达式.

No20 实现一个函数来判断字符串是否表示数值。

```
return string.matches("[\\+\\-]?\\d*(\\.\\d+)?([eE][\\+\\-]?\\d+)?");
```

No20-2 将字符串转换成整数；不合法返回 0；

从头开始遍历，先看是否为+-，标记正负号。然后一直遍历，如果出现为的字符在 0-9 之间。

```
for(int i=index;i<str.length();i++){
    if(str.charAt(i) - '0' > 9 || str.charAt(i)-'0' < 0) return 0;
    sum = result * 10 + str.charAt(i)-'0';
    result = sum;
}
return symbol*result;
```

累积相加，并判断是否越界。

No21,调整数组顺序使得奇数位于偶数之前。冒泡排序；if(array[j]%2==0 && array[j+1]%2==1){交换}

No22,返回链表的倒数第 k 个节点。

快慢指针，快的走 k 个，然后慢快一起，到尾。

No23,链表中环的入口节点;

从头开始，快慢节点，如果相遇，那就有环。不相遇就没有环。相遇之后，快的从头出发，一起走，相遇的地方就是入口节点。

```
ListNode fast = pHead,slow = pHead;
while(fast.next!=null){ fast =fast.next.next;slow =slow.next; if(fast == slow) break;}
if(fast == null)
    return null;
fast = pHead;
while(fast != slow){
    fast = fast.next;
    slow = slow.next;
}
return fast;
```

No24,反转链表,并返回反转之后的头结点。头插法；while(cur.next!=null){将 cur.next 插入 dummy 后面}；

No25,合并两个有序的链表，重建一个新的，包含所有的两个链表。--->单调不减 l1.next = merge(l1.next,l2)

No26,输入 A 和 B 两个树，判断 B 是 A 的子树。空树不是子树 先看是否马上包含

No27,二叉树的镜像。输入一个二叉树，将他变成他的镜像 递归，交换左右子树，然后递归左右子树。

No28,输入一个二叉树，判断二叉树是不是对称的。

主要看左子树和右子树是否对称

```
helper(root1.left,root2.right) && helper(root1.right,root2.left);
```

No29,顺时针打印矩阵。

```
        while(rowStart<=rowEnd && colStart <= colEnd)
            helper(resList,matrix,rowStart++,rowEnd--,colStart++,colEnd--);
```

No30,包含 min 函数的栈 一个正常栈，一个 min 函数栈。

No31,栈的压入弹出序列,第一个表示压入序列，判断第二个是不 是弹出序列。

维护 j 表示弹出序列的位置，将压入序列压入栈中，如果栈顶和弹出的一样，弹出。最后如果栈为空，T.

```
        for(int j=0;j<pushA.length;j++){
            stack.push(pushA[j]);
            while(!stack.isEmpty() && stack.peek() == popA[i]){
                stack.pop();
                i++;
            }
        }
```

No32,从上到下打印二叉树 创建队列，先根入队，然后当队非空{弹出，操作，将左右子树入队}

之字形打印二叉树

```
        LinkedList<TreeNode> queue = new LinkedList<>();
            queue.add(pRoot);
            TreeNode cur = null;
            boolean flag = true;
            while (!queue.isEmpty()){
                int size = queue.size();
                ArrayList<Integer> list = new ArrayList<>();
                for (int i=0;i<size;i++){
                    cur = queue.poll();
                    if(cur.left!=null)
                        queue.add(cur.left);
                    if(cur.right!=null)
                        queue.add(cur.right);
                    list.add(cur.val);
                }
                if(!flag)
                    Collections.reverse(list);
                resList.add(list);
                flag = !flag;
        }
        return resList;
```

No33,输入一个整数数组，判断是不是二叉搜索树的后序遍历序列

No34,二叉树中和未某一值得路径

```
public void helper(ArrayList<ArrayList<Integer>> resList,ArrayList<Integer> tempList,TreeNode root,int target){
        if(root == null || root.val > target)    return;
        tempList.add(root.val);
        if(root.val == target && root.left == null && root.right==null)
            resList.add(new ArrayList(tempList));
```

```
            helper(resList,tempList,root.left,target-root.val);
            helper(resList,tempList,root.right,target-root.val);
            tempList.remove(tempList.size()-1);
}
```

No35 复杂链表的复制;链表不仅有 val,next,还有一个指向任意节点

```
    Map<RandomListNode,RandomListNode> map = new HashMap<>();
    创建一个 map,key，value 都是节点。将原本的链表节点压入为 key,value 为值一样的链表；
    将
    map.put(cur,new RandomListNode(cur.label));
    cur = pHead;
        while (cur != null){
            map.get(cur).next = map.get(cur.next);
            map.get(cur).random = map.get(cur.random);
            cur = cur.next;
        }
        return map.get(pHead);
```

No36 二叉搜索树和双向链表;二叉搜索树转换成一个排序的双向链表，不能创建新的节点，只能调整指针顺序。中序遍历，保持头结点，以及 pre 节点，从栈中弹出的节点 cur 和 pre 建立联系。

```
                cur = stack.pop();
                if(realHead==null){
                    realHead = cur;
                    temphead = cur;
                }else{
                    temphead.right = cur;
                    cur.left = temphead;
                    temphead = cur;
                }
                cur = cur.right;
```

No37 实现函数序列化二叉树和反序列二叉树

```
public class Solution {
    public int index = -1;
    String Serialize(TreeNode root) {
        StringBuffer sb = new StringBuffer();
        if(root == null){
            sb.append("#,");
            return sb.toString();
        }
        sb.append(root.val + ",");
        sb.append(Serialize(root.left));
        sb.append(Serialize(root.right));
        return sb.toString();
    }
    TreeNode Deserialize(String str) {
```

```
            index++;
           int len = str.length();
            if(index >= len){
                return null;
            }
            String[] strr = str.split(",");
            TreeNode node = null;
            if(!strr[index].equals("#")){
                node = new TreeNode(Integer.valueOf(strr[index]));
                node.left = Deserialize(str);
                node.right = Deserialize(str);
            }

            return node;
    }
}
```

No38 输入一个字符串，给出所有的排列;

首先这是个排列问题，每次遍历从字符串取值的时候，都要从 0 开始，且维护一个 boolean 数组表示第 i 位是否已经选中。

其次，我们要进行去重判断，也就是 `if (used[i] || (i >0 && str[i] == str[i-1] && !used[i-1])) continue;`

```
            if(sb.length() == str.length){
                resList.add(sb.toString());
                return;
            }
            for (int i=0;i<str.length;i++){
                if (used[i] || (i >0 && str[i] == str[i-1] && !used[i-1])) continue;
                used[i] = true;
                sb.append(str[i]);
                helper38(resList,sb,str,used);
                used[i] = false;
                sb.deleteCharAt(sb.length()-1);
            }
```

No39,数组中出现超过一半的数字,如果不存在，返回 0；

摩尔投票：int count =0,result = 0;遍历数组，如果等于 result，count++,否则如果 count==0,换人。否则 count--;最后再遍历看等于 reslut 的人有几个，是否过了一半，

No40，最小的 k 个数，然后先进去 K 个，如果比较和堆顶大小，符合条件出来一个，再进去

首先建立一个大小为 K 的优先队列最大堆，默认最小堆。

```
            PriorityQueue<Integer> maxHeap = new PriorityQueue<>(k, new Comparator<Integer>() {
                @Override
                public int compare(Integer o1,Integer o2){
                    return o2.compareTo(o1);
                }
            });
            for(int i=0;i<input.length;i++){
```

```
            if(maxHeap.size()!=k){
                maxHeap.offer(input[i]);
            }else if(maxHeap.peek()>input[i]){
                    maxHeap.poll();
                    maxHeap.offer(input[i]);
                }
        }
```

No41，数据流中的中位数，奇数个，中间那个。偶尔个，中间俩的平均

No42,连续子数组的最大和，数组中有正数有负数，求所有连续子数组的最大和。

```
dp[i] = Math.max(dp[i-1]+array[i],array[i]);
```

No43,1-n 所有数中，求所有十进制位出现 1 个总数

No44，从 0-n,所有的从前到后排到一块，实现一个函数，求第 k 位的数字是几？

No45,给定一个数组，求组合到一块的最小数字

首先将整数数组变成字符串数组，然后排序，排序方法就是 o1+o2.compareTo(o2+o1)

```
String[] strs = new String[numbers.length];
for(int i=0;i<numbers.length;i++)
    strs[i] = String.valueOf(numbers[i]);
Arrays.sort(strs,new Comparator<String>(){
    @Override
    public int compare(String o1,String o2){
        return (o1+o2).compareTo(o2+o1); //默认都是升序
    }
});
  String res = String.join("",strs);
```

No46,0-25 翻译成 a-z,给一个数字，求有几种翻译方法

No47,礼物的最大价值，矩阵，每一步都有一个值，从左上到右下，只能向右或向下移动，求最大和

No48，最长不含重复字符的子字符串，

No49,只含 2,3,5 因子的数是丑数，1 也是，求第 n 个丑数

这个是非常规动态规划：维持 235 的因子个数 i,j,k

丑数的排序：index[1] = 1,第一个丑数，

```
int i=j=k=1;
for(int m=2;m<=n;m++){
    dp[i] = min(dp[i]*2,dp[j]*3,dp[k]*5);
    if(dp[m]= dp[i]*2) i++;
if(dp[m] == dp[j]*3) j++;
if(dp[m] == dp[k]*5) k++;
}
```

No50，第一个只出现一次的字符,返回的是位置

```
int[] nums = new int[128];
for(int i=0;i<str.length();i++){
    nums[str.charAt(i)]++;
}
```

然后遍历，找到第一个值为 1 的。

No51，数组中的逆序对,归并排序！！！！

归并的时候，从大到小比较，如果左边的比右面的大，那么 count += right – mid +1;

No52,两个链表的第一个公共节点。

要么算两个长度，然后长的走两步，然后同时走，走到遇见相等的。

要么 l1 走完走 l2,l2 走完走 l1。如果存在肯定有相同的。

```
while(l1 != l2){
    l1 = (l1 == null) ? headB : l1.next;
    l2 = (l2 == null) ? headA : l2.next;
}
return l1;
```

如果是比较两个链表是否有交点，直接看俩链表的最后一个节点是否一样。

No53,在排序数组中查找数组，一个数字出现了多少次

```
public int getLeft(int[] nums, int k){
    int left = 0;int right = nums.length-1;int res = -1;
    while(left <= right){
    int mid = left+(right-left)/2;
     if(nums[mid] == k ) res = mid;
     if(nums[mid] >= target) right =mid-1;
     else left =mid+1;}
}
```

No54,二叉搜索树中，第 K 小节点

No55-1，二叉树深度　递归，空为 0，求左子树，求右子树。返回左右最大的+1；

No55-2,是不是平衡二叉树　递归求深度，然后加一步，左的深度-右的深度绝对值是否大于 1；

No56-1,一个数组，一个出现一次，其他出现两次，求一次的。

```
int diff =0;
for(int n : array){
    diff = diff^n;
}
diff =diff & -diff;
for(int n:array){
    if((diff & n) == 0){
        num1[0] = num1[0]^n;
    }else{
        num2[0] = num2[0]^n;
    }
}
```

No56-2,数组，一个出现一次，其他出现三次。求一次

No57-1，递增排序的数组，和 target.找出一对和为 target 的数字

双指针，一个头一个尾，看结果移动。

No57-2,正数 s,给出连续正数序列，其和未 s.所有的序列。序列最少有俩。

```
双指针，如果区间比 target 大，small++,f 否则 big++;
int small =1,big =2;
int mid = (1 + sum) / 2;
while ( small <= mid){
    int cur = (small+big) * (big - small +1)/2;
```

```
                if (cur > sum)
                    small++;
                else if (cur < sum)
                    big++;
                else {将这个结果加入最终的结果集中}
```

No58-1,翻转单词顺序

用 API 快点，

```
public String ReverseSentence(String str) {
    if(str==null||str.trim().equals(""))// trim 掉多余空格
        return str;
    String[] words = str.split(" ");// 以空格切分出各个单词
    还他妈不如对每个字符串进行翻转呢。先以字符串建 StringBuffer,然后 reverse()方法；
    strs.join(“ ”, words);
}
```

No58-2,左旋转字符串，三次翻转。

```
        swap(chas,0,len-1);
        n = n % len;
        swap(chas,0,len-n-1);
        swap(chas,len-n,len-1);
```

No59-1 给出数组和窗口大小，求滑动窗口的最大值。

No59-2,实现带有 max 函数的队列，出入队和 max 都是 O(1)

No60,n 个色子，点数和为 s，s 所有可能出现的值得概率

No61,扑克牌抽 5 个数，是不是顺子。大小王可以为任何;

```
    int[] map = new int[14];遍历这五个数，如果是大小王，不理他们。如果出现重复了，直接返回 false.
    同时遍历过程中，维护一个最大值一个最小值。最后看看 max-min>4?
```

No62,0 到 n-1,围一圈，从 0 开始，删除第 m 个数字。求最后的数字；

```
    维持几个数，boolean[] falg =new boolan[n];表示是否存活。
    count = n,表示存活个数。 step = 0 表示这一轮的第几个人, inde=-1 表示遍历数组的位置。
while(count > 0){
    index ++;
    if(index == n) index =0;
    if(flag[index]) continue; //遍历直到找到还健在的人
    step++;if(step == m){ count--;flag[i] = true; step = 0;}}
```

No63,数组，值为股票当时的价格，求最大利润

No64,求 1+2+3+...+n,不能乘除，for,while,if,else 等。

1.需利用逻辑与的短路特性实现递归终止。 2.当 n==0 时，(n>0)&&((sum+=Sum_Solution(n-1))>0)只执行前面的判断，为 false，然后直接返回 0；3.当 n>0 时，执行 sum+=Sum_Solution(n-1)，实现递归计算 Sum_Solution(n)。

```
    public int Sum_Solution(int n) {
        int sum = n;
        boolean ans = (n>0)&&((sum+=Sum_Solution(n-1))>0);
        return sum;
    }
```

No65,不用加减乘除做加法

第一步：相加各位的值，不算进位，得到 010，二进制每位相加就相当于各位做异或操作，101^111。

第二步：计算进位值，得到 1010，相当于各位做与操作得到 101，再向左移一位得到 1010，(101&111)<<1。

```
    public int Add(int num1,int num2) {
        while(num2 != 0){
            int temp = num1;
            num1 = num1^num2;
            num2 = (temp & num2)<<1;
        }
        return num1;
    }
```

No66，构建乘积数组

先创建数组，从头遍历 A,每个 B[i]保存前 i-1 个数的乘积。

然后从尾到头遍历数组 A,用 temp 保存 i 后面的乘积。然后 B[i] = temp * B[i],更新 temp;

```
        B[0] = 1;
        for(int i=1;i<A.length;i++){
            B[i] = B[i-1] *A[i-1];
        }
        int temp=1;
        for(int i=A.length-1;i>=0;i--){
            B[i] = B[i] * temp;
            temp = A[i] * temp;
        }
        return B;
```

字符流中第一个不重复的数：用 StringBuffer 老保存字符，用 128 个 map 保存每个字符的个数。让之前保存的数组，从头开始遍历，找到第一个个数统计为 1 的值。

```
    StringBuffer sb = new StringBuffer();
    int[] map = new int[256];
    public void Insert(char ch)
    {
        sb.append(ch);
        map[ch]++;
    }
    public char FirstAppearingOnce()
    {
        char[] chars = sb.toString().toCharArray();
        for(char c : chars){
            if(map[c] == 1)
                return c;
        }
        return '#';
    }
```

1.栈排序

申请一个辅助栈，将数据栈弹出，如果辅助栈为空或者 cur>=辅助栈元素，直接压入辅助栈。

否则的话，辅助栈弹出，并压入数据栈。直到 cur >=辅助栈顶。

2.链表实现队列

入队进入列表，出队，去除头结点，返回头结点值。

3.最长连续递增序列

If(nums[j] > nums[j-1]) dp[j] = dp[j-1]+1;

4.最长不连续序列

遍历序列 i,同时维持 j,和 set,如果能放入就好，不能的话，就慢慢将 j 一个个移除。

5.二维数组回行打印

还是那一套。

6.无序数组构建一棵二叉排序树,先排序，再构建。递归构建；

7.一个数组实现两个栈

维护 size=0，size2 = length-1;栈 1 进栈，size++;出栈--，设为 null;

栈 2 进栈，size--,出栈，++；

8.二叉树宽度 层序遍历，求每层的 size 最大

9.二叉树是否对称， helper(root.left,root.right);

10.链表 m 到 n 反转 ，双指针，快的先走 m 步，头插法将 m 后面的插入再走 n-m 次。

11.一个 n 位数，现在可以删除其中任意 k 位，使得剩下的数最小

12.实现有符号大数链表加法，靠近头结点位置为高位

13.字符串横向改纵向

14.八皇后的问题

15.找出来数组中每个元素后边第一个比它大的值

16.给你一个二叉树，从上往下看，然后左往右顺序输出你能看到节点，同一个竖直方向上上面的节点把下面的节点遮挡住了

17.链表反转，分别用遍历与递归实现

```
public Node reverNode(Node head){
    if (head == null || head.next == null)
        return head;
    Node newHead = reverNode(head.next);
    head.next.next = head;
    head.next = null;
    return newHead;
}
```

18.完全二叉树的最大深度与节点个数 //求根节点左子树高度，和右子树高度，如果一致，左子树满。左子树加上根节点 2^left,然后递归求右子树的个数。如果不一致，右子树满，然后公式求右子树，然后递归左子树。

```
public int getWanQuan(TreeNode root){
    if (root == null)
        return 0;
    TreeNode left = root.left;
    int lcount = 0;
    while (left != null){
        lcount++;
```

```
            left = left.left;
        }
        int rcount = 0;
        TreeNode right = root.right;
        while (right != null){
            rcount++;
            right = right.right;???? //  应该为 left,这个算法还是不如上一个清晰。
        }
        int res;
        //左子树和右子树高度相同，左子树是满的，右子树再递归求点数
        if (lcount == rcount){
            res = (int)Math.pow(2,lcount) + getWanQuan(root.right);
        }else {
            res = (int)Math.pow(2,rcount) + getWanQuan(root.left);
        }
        return res;
    }
```

19.两个栈实现队列 入队进栈，出队的时候，如果出栈不空就出，否则入栈的所有都倒入出栈，然后出。

20.两个有序数组交集、并集

新数组指针 k=0;

两个数组分别进行遍历 i=0,j=0.如果相等，num[k++] = a[i++],j++;如果不等，a 小，i++,否则，j++;

并集：如果不等，小的入 K++,如果相等，入 k++,两人都走。

21.给定一个有序存在重复的值链表，使得每个元素只出现一次

删除链表，pre 和 cur,如果 cur 和 pre 一样了，删除 cur,如果不一样，同步走。

22.leetcode 200

```
    private int m,n;
    private int[][] directions = {{0,1},{0,-1},{1,0},{-1,0}};
    public int numIslands(char[][] grid) {
        if(grid == null ||grid.length == 0)
            return 0;
        m = grid.length;
        n = grid[0].length;
        int result =0;
        for(int i =0;i<m;i++){
            for(int j =0;j<n;j++){
                if(grid[i][j] == '1'){
                    dfs(grid,i,j);
                    result++;
                }

            }
        }
        return result;
    }
```

```
public void dfs(char[][] grid,int i,int j){
    if(i <0 || j<0 || i >= m || j >=n || grid[i][j] == '0')
        return;
    grid[i][j] = '0';
    for(int k=0;k<directions.length;k++){
        dfs(grid,i+directions[k][0],j+directions[k][1]);
    }   }
```

23.二叉搜索树转有序双向链表(中序遍历)

24.字符串全排列 ，可能有重复的，要去重

```
public void helper38(ArrayList<String> resList, StringBuffer sb,char[]  str,boolean[] used){
    if(sb.length() == str.length){
        resList.add(sb.toString());
        return;
    }
    for (int i=0;i<str.length;i++){
        if (used[i] || (i>0 && str[i] == str[i-1] && !used[i-1])) continue;
        used[i] = true;
        sb.append(str[i]);
        helper38(resList,sb,str,used);
        used[i] = false;
        sb.deleteCharAt(sb.length()-1);
    }
}
```

25.二叉搜索树第 k 个节点，不用中序遍历，还是中序遍历吧，非递归。

26.有序数组查找重复元素个数

二分查找：先求左边的， 再求右边的。

```
While(left <right){
Mid = left+(right-left)/2;
If(nums[mid] == target) res = mid;
If(nums[mid] >= target) right = mid-1;
Else left = mid+1;
}
Return res;
```

27.定长数组实现队列

28.用二分法对一个数字开根号

29.判断一颗树是不是二叉搜索树（中序遍历是否有序；记录 pre，和当前比较，是否 pre 一直小于当前）

30.Excel 表的列字母转换，输入第几列，输出列字母组合

31.链表第 k-1 个节点

32.手撕快排

33.二分查找

34.一个无序有正有负数组，求乘积最大的三个数的乘积

35.求二叉树的深度，不使用递归 ：层序遍历和递归

36.实现链表，无序链表，对链表值奇偶分离并排序，空间复杂度 O(1)

37.单调不递减数组，给一个 target，找出大于等于 target 的下标 index

38.单调不递减链表，删除掉重复值

39.无序数组构建一棵二叉排序树(先排序，然后取中间的节点生成根节点，左边的生成左子树，右边的右子树)

40.行和列都是有序的二维矩阵找一个 target 值

得到 mid 之后，要算出他在哪行哪列。

41.是否是回文链表(快慢指针，找到中间节点，然后后面的入栈，弹出、遍历对比)；

42.打印出根节点到叶子节点的最长路径

43.双链表按照奇偶顺序分成两个链表，要求不要复制链表

44.不严格递增数组，要求删除出现次数大于 k 的数字，要求不要新建存储空间

还是维持一个符合条件的指针的 j;

45.链表相邻元素交换

两个一换，while（cur!=null && cur.next != null）,调整 pre,cur,cur.next,cur.next.next 之间的 cur 和 next 的顺序。

46.二叉树的最小公共祖先

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    if(root == null || root == p || root == q)
        return root;
    TreeNode left = lowestCommonAncestor(root.left,p,q);
    TreeNode right = lowestCommonAncestor(root.right,p,q);
    return (left!=null && right!=null)?root:(left == null ? right : left);
}
```

47.字符串形式自定义进制大数相加

48.链表每隔 k 个反转

要么用 pre,start,end,next。Cur 每走一步，计算，到 count 时，pre,pre.next,cur,cur.next 做链表翻转。然后 count 清零。继续遍历 cur,知道 cur 为空。

要么，先走一遍，得到长度。然后 lenht> k，也用 pre,pre.next,cur,cur.next 做链表翻转。然后 len = len-k;

49.输出根节点到叶子节点路径之和为 target 的路径列表（helper(resList,tempList,root,sum);

50.一些数，任意排列求可形成最小的值

```
public String PrintMinNumber(int [] numbers) {
    String[] strs = new String[numbers.length];
    for(int i=0;i<numbers.length;i++)
        strs[i] = String.valueOf(numbers[i]);
    Arrays.sort(strs,new Comparator<String>(){
        @Override
        public int compare(String o1,String o2){
            return (o1+o2).compareTo(o2+o1);
        }
    });
    String res = String.join("",strs);
    return res;
}
```

51.LeetCode　1038.

52.数组题，任意一个整型数组，判断是否可以将数组分为三个区间，每个区间中数值的和相同

53.已排序的整数数组去重

一个遍历指针，一个去重之后的位置指针。如果重就++，不重就将当前遍历放入去重指针 j++;

54.错位的全排列（第一位不能是 1，第二位不能是 2）

55.k 路链表归并

```
public ListNode mergeKLists(List<ListNode> lists) {
    if (lists==null||lists.size()==0) return null;

    PriorityQueue<ListNode> queue= new PriorityQueue<ListNode>(lists.size(),new
Comparator<ListNode>(){
        @Override
        public int compare(ListNode o1,ListNode o2){
            if (o1.val<o2.val)
                return -1;
            else if (o1.val==o2.val)
                return 0;
            else
                return 1;
        }
    });

    ListNode dummy = new ListNode(0);
    ListNode tail=dummy;

    for (ListNode node:lists)
        if (node!=null)
            queue.add(node);

    while (!queue.isEmpty()){
        tail.next=queue.poll();
        tail=tail.next;

        if (tail.next!=null)
            queue.add(tail.next);
    }
    return dummy.next;
}
```

56.非降序数组，找与 target 最相近的数的下标

57.二叉树逆时针打印最外层节点

先左边排到第二，然后先序或中序得到叶子节点，然后右边的跟二到倒二。

58.输入一个数字 n，构建一个完全二叉树并输出、

层序遍历构建二叉树。

59.输入一个矩阵，起始点和目标点，判断是否存在可达路径

60.无向图最短路径

61.第 K 层叶子节点个数 （层序遍历，level = k，弹出 k 层的节点，统计叶节点个数）

迪杰斯特拉(Dijkstra)， **Dijkstra 算法可以计算任意节点到其他节点的最短路径**

贪心策略是每次选可达的点中距离源点最近的点进行扩展，即贪心选取最短距离的点

指定一个节点，例如我们要计算 'A' 到其他节点的最短路径

引入两个集合（S、U），**S 集合包含已求出的最短路径的点**（以及相应的最短长度），**U 集合包含未求出最短路径的点**（以及 A 到该点的路径，注意 如上图所示，**A->C 由于没有直接相连 初始时为∞**）

初始化两个集合，**S 集合初始时 只有当前要计算的节点，A->A = 0，**

U 集合初始时为 A->B = 4, A->C = ∞, A->D = 2, A->E = ∞，敲黑板！！！接下来要进行核心两步骤了

从 U 集合中找出路径最短的点，加入 S 集合，例如 A->D = 2

更新 U 集合路径，if ('D 到 B,C,E 的距离' + 'AD 距离' < 'A 到 B,C,E 的距离') 则更新 U

循环执行 4、5 两步骤，直至遍历结束，得到 A 到其他节点的最短路径

库鲁斯卡尔(Kruskal) :每次选择最小权的边，加入树中，但不能生成环。

贪心策略是每次选最短的边（刨除成环的边）来作为最小生成树，**即贪心最短边**

普里姆算法(Prim)，从选中的一个点开始，找到和这些点相连的其他集合的边中选取最短的那条，然后将那个点加入已选集合。**从已选点集合中，选取和已选集合边最小的点，加入集合。**

贪心策略是每次选可达的点中距离曾经扩展过的点中任意点的最短距离，类似 Dij，只是不是找距离源点的最短距离

KMP

贪心策略 0.0 不是贪心

是动态规划，动态规划的是当前状态失败之后上一次匹配的位置（求的是最长的与前缀子串匹配的左子串）

1. Two Sum:给定数组和 target,返回两个元素之和为 target 的元素 index;

遍历数组时，将 target-当前元素，index 放入 map。如果存在结果，map.get(nums[i])!=null，说明已经合格了。

2. Add Two Numbers：他是一个反转的链表，求真正链表代表数字的和的链表的翻转；

其实就是俩链表一一相加，进位就进位。

其中技巧是 while(l1 != null || l2 != null){cur1 = l1==null ? 0: l1.val; l1 = l1 == null ? null :l1.next};

字符串 **3. Longest Substring Without Repeating Characters：字符串最长的不重复子序列长度：**

遍历字符串，同时又整个 j 指针，维持个 set，将当前字符，放入 set,如果放不进去，就是有重复的，j 位置的支付删除，直到放进去。这时候，以 i 为结尾的不重复序列长度就是 i-j+1;然后对出个最大的。

```
While(set.contains(c)){
    Set.remove(str.charAt(j++));
} set.add(c);   max = Math.max(max,j-i+1);
```

字符串 5. Longest Palindromic Substring：字符串最长的回文序列。

从头到尾遍历字符串：对遍历的进行从里到外扩展，扩展有两个形式，一是 expend(i,i),二是 expend(i,i+1)，**扩展过程中，直到碰到不回文的。然后计算出回文的最大长度和 left;**

```
void  Expend(String str, int left, int right){
    while(left >=0 &&right <str.length()    && str.charAt(left) == str.charAt(right) )
    left--;right++;}
```

if(max < right-left-1){

max = right-left-1; leftMin = left+1;

}

字符串 6. ZigZag Conversion：一个字符串，一个行数。然后字符竖折竖折来排序。然后一层一层的组合字符。

M 行：**整个 StringBuffer[m]数组**;遍历字符串，先向下走 m 个，每个分到一个 sb 中，然后上反走 m-2 个，每个分到一个 sb 中。完事之后，m 个 sb 都加到 sb[0]之后，然后 toString();

7. Reverse Integer：将整数值反转，如 120---21，-21---- -12，如果范围大于整数范围，就返回 0；

为了防止溢出，long = result; 对输入的整数值 x,循环到 x 不等于 0：**result = result *10 + x %10;X = x/10;同时判断 result 是否溢出，大于 Integer.MAX_VALUE 小于 MIN_VALUE。即对 x 不断取余，获取个位数。然后作为 result 的高位。在这个过程中正负无所谓**。-123%10=-3;-123/10=-12;

8. String to Integer (atoi)：前面的空字符不理，非空之后，如果遇到非 s 数字，直接不再搭理后面的，结果定型。如果遇到+-计正负，然后遇到整数就记结果,直到结束或者遇到非数字。

首先，去除前面的空。然后，对第一个字符进行是否+-判断，标记住。然后，继续遍历，先看是否非数字，还是如何判断是否越界的问题，如果是数字，就一步步高位取结果。如果遇到非数字，直接跳出，返回结果。

判断是否越界，可以用 long result = 0;每位遍历之后，**看 result 是否大于 int 最大值和 int 最小值。**

如果是，则越界。返回特殊处理，否则继续进行。

9. Palindrome Number，一个数，正反来是否一样。负数不一样

当 x<0，不回文。然后反转这个树，取余作高位。看看这俩结果是否一样。

12. Integer to Roman

```
public String intToRoman(int num) {
        String M[] = {"", "M", "MM", "MMM"};
        String C[] = {"", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"};
        String X[] = {"", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC"};
        String I[] = {"", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX"};
        return M[num/1000] + C[(num%1000)/100] + X[(num%100)/10] + I[num%10];
    }
```

14. Longest Common Prefix；N 个字符串，求这 N 个字符串最长公共前缀

把第一个字符串当做前缀。遍历这剩下来的字符串：如果 while(strs[i].indexOf(pre)!=0),说明 pre 减去一位，substing(0,length()-1);或者，一个个 I,每个字符串遍历看是否和第一个一致。如果一直，i++。不相等就 break;

```
        String pre = strs[0];
        for(int i=1;i<strs.length;i++){
            while(strs[i].indexOf(pre) != 0)
                pre = pre.substring(0,pre.length()-1);
        }
        return pre;
```

15. **3Sum,数组中，找 3 个和为 0 为三个数，每个数不能用两次。要考虑重复的问题。**

先排序。然后从 0 到 n-3 遍历。Sum=0-num[i];然后进行二分查找，一头 i+1,一尾。如果等于 sum，结果保存，同时如果左边的有重复的，一直++，如果右边的重复的，一直--。然后再 left++,right--;如果不等于然后调整 left 和 right;

```
For(int i=0;i<len-2;i++){
If(i>0 && nums[i-1] == nums[i] ) continue;
Int left = i+1,right = len-1,sum = -nums[i];
While(left <right){
Int temp = nums[left] + nums[right];
If(temp == sum){
resList.add(Arrays.asList(nums[i],nums[left],nums[right]));
while(left<right && nums[left+1] == nums[left]) left++;
while(left<right && nums[right-1]==nums[right-1]) right--;
left++;right--;
}else if( temp < sum) lef++;else right--;
}
}
```

16. 3Sum Closest:一个数组，找出 3 个和最进阶 target 的。

首先，排序。然后先随便三个数相加得到 res;开始遍历，从 0 到 n-2,然后对每个 i，left=i+1,right 为尾部。While(left<right){sum=这三个相加,根据 sum 和 target 的大小调整两个指针，res 和 target 的差与 sum 与 target 的差距保存最小的.}

17. Letter Combinations of a Phone Number

```
Public void combation(List<String> resList, String prefix,int offset, String digits){
If(offset == digits.length()){
    resList.add(prefix);return}
String letters = keys[digits.charAt(offset)-'0'];
For(char c:letters.toCharArray()){//相当于遍历一下
    Combation(resList,prefix+c,offset+1,digits);}
}
```

18. 4Sum 数组中 4 个数之和为 0，考虑重复的数，但一个数不能用两次，组合也不能太一致。

先排序。然后遍历第一层，如果重复 continue，然后遍历第二层：继续考虑重复。Sum = target-那俩。然后双指针，如果等于 sum,加入，然后去重。左右指针移动，不等的话就根据大小调整。

19,删除倒数第 k 个节点

Dummy,然后 fast 走 k 步，且 fast 不能为 null,如果走了 k 步，就说明长度大于 k,还可以删。然后 low 和 fast 一块走，fast 到末尾。

20、**括号序列是否匹配**:

弄个栈，如果遇见左括号类型的，就将该类型的右括号入栈。如果是右括号，弹出看是否一致。若一致且最后为为空，返回 true：

21、合并两个有序序列；

非递归，两个链表，一次比较入队。然后剩余的入队。

递归，链表为空，返回另外一个。然后比较俩链表大小，小的 l1.next = mer(l1.next,l2);

22、n 对小括号的全正确排序：

正常的排列组合问题。这是两个变量，一个 left,一个 right。辅助函数是 resList,string 代替 tempList,左括号数量和右括号数量；为了使得左括号始终再右括号前面，辅助函数要走两边，第一遍加（，第二遍加）；

```
public void helper(List<String> resList,int left,int right, String s){
```

```
        if(left==0&&right==0){
            resList.add(s);
            return;
        }
        if(left>0)
            helper(resList,left-1,right+1,s+"(");
        if(right>0)
            helper(resList,left,right-1,s+")");
    }
```

24：成对的翻转链表

while(pre.next!=null && pre.next.next!=null){

将 cur 后的 next 利用头插法出入到 pre 的后面。然后 pre 和 cur 再后移；

}

Dummy;当 dummy 的 next 和 next.next 都不为空时；cur 作为 pre，next 作为第一个节点，next.next 作为第二个节点，然后将第二个节点插入到 pre 的后面。然后 cur 移动两个节点。

26、有序数组，删除多余的重复数组，in-place 删除，然后返回长度。

遍历数组，再加一个不重复的指 j 针。当 num[j] =当前的时候，继续遍历，不等于时，将当前的移动到 nums[j]并加加；

27、数组，删除指定的数字；

同上。遍历数组，并维持一个处理后指针；

28、**实现 indexOf(String s1,String s2);**

往死里遍历：外层遍历：从 0-（s1 的长-s2 的长）：里层遍历：从 0 到 s2 的长度，如果出现了不一致，break;如果 j 走到了最后完全一致，就可以直接返回了。

33、左移一部分的有序数组中，查询 target

二分查找的变形；

34、有序数组中，查询一个 target 第一次和最后一次出现的位置；

二分查找的变形。

35、将一个数插入到有序数组中，求位置

二分查找

39：非重复数组，找出和为 target 的数组，一个不能用两次；

排列组合类似的回溯问题；

40，带有重复的数组，找出和未 target 的数组组合，一个不能用两次；

回溯+组合。

43，两字符串的数字，相乘，得到结果，输出字符串。

一个字符串的 indexI 为 0-n-1,一个字符串的 indexJ 为 0-n-1;这两个相乘的位置是放在 index[i+j,i+j+1];

46,不重复数组的，所有全排列；

回溯+排列；

47，带有重复的数组，全排列；

回溯+排列；

48：n x n 矩阵，顺时针转动 90 度。

和那个顺时针打印矩阵一样，这个也是一圈圈的转动，先外圈，再内圈。Temp+置换的形式。

49，一群字符串，有些字符串字母组成一样，有的不一样，将一样的放一块；

遍历数组：将字符串转化成数组，然后排序。将排序的作为 key,真正的值作为放入 list 作为 value; 这样 hashmap 的 values 就是结果；

```
public List<List<String>> groupAnagrams(String[] strs) {
        List<List<String>> resList = new ArrayList<>();
        Map<String,List<String>> map = new HashMap<>();
        for(String str: strs){
            char[] chas =str.toCharArray();
            Arrays.sort(chas);
            String key = String.valueOf(chas);
            if(!map.containsKey(key))
                map.put(key,new ArrayList<>());
            map.get(key).add(str);
        }
        return new ArrayList<>(map.values());
    }
```

50：计算 power(x,n); x 在正负 100，n 在正负 2^31;

```
public double myPow(double x, int n) {
        if(n == 0)
            return 1;
        if(n == Integer.MIN_VALUE)
            return myPow(x*x, n/2);
        if(n < 0){
            x = 1/x;
            n = -n;
        }
        return n % 2 == 0 ? myPow(x*x,n/2) : x * myPow(x*x,n/2);
    }
}
```

54:一圈圈顺时针打印数组。

先整外圈，再里圈。

58：最后一个单词的长度

从后遍历，计数，知道遇见空格。

59：给一个 N,将 1-N^2,整成上面外圈顺时针到里圈的矩阵。

差不多。先建立 NN 数组，然后从外圈到里圈遍历：每圈的时候，顺时针赋值。

60：1-N 这 n 个数字的全排列，求第 K 小的；

主流思想我一下子理解不了啊

61、翻转链表，链表右移 K 为，可以循环多移；

第一、首先遍历链表 fast,得到链表的长度。然后整 slow 指针，移动 len-K 个，然后将后面的 k 个移动到 dummy 的后面，fast 作为最后的指针指向本来的头结点，然后 slow 指向 null;

62、矩阵移动，mxn 矩阵从左上角移动到右下角，只能横移和竖移，共有多少方法；

第一行只有一个方法， 第一列只有一种。然后 dp[i][j] = dp[i-1][j] + dp[i][j-1];

63、矩阵移动，从左上移动到右下：横竖移动，矩阵值为 1 的不能走；求多少种移动方式。

同上，只不过在判断的时候，先判断值是否为 1，为 1 表示 0；然后其他的一样.

64、矩阵，从左上到右下，最短的路径和多少：

双层遍历，都为 0 为等于左上角。一行和一列，等于前面的那个加自身。其他的，等于上面和左边最小的那个加自身。

66、数组表示的数+1，然后返回之后的

数组从后遍历：如果数字不为 9，直接加，然后返回原数组。如果为 9，设为 0，然后下一个加一。如果最后一只都是 999，然后新建数组，长度加 1，初始为 1；

67、两个二进制字符串相加，得到的二进制字符串

两个字符串，只要有一个不空，从尾到头相加，加的结果封二进 1，整一个进位标记 carry，每位的加人 StringBuffer,最后有进位，咱也加入。最后再反转。

69. Sqrt(x)　实现 int sqrt(int x)；

```
    long r = x;
while (r*r > x)
    r = (r + x/r) / 2;
return (int) r;
```

70、n 层楼梯，爬上去的方式有几种？只能一层或两层。

Fn = Fn-1 + Fn-2;

73、mxn 的数组，如果有一个为 0，那么就让他的所有列和行都为 0；in-place,求最后的数组；

双层遍历遍历数组，如果值为 0，横向第一列为 0，竖向的第一行为 0；然后有第一行或者第一列为 0，比较下 fc,fr 为 0。然后再次遍历第一行和第一列的，如果遇见 0，再竖向和横向的全设为 0。如果存在 fc 和 fc 为 0，一行和一列全设置为 0；

74、mxn 的矩阵，层序遍历就是有序的数组。在矩阵中搜索 target;

二分查找；

75：一个 2/1/0 的数组，将 0 分到左边，2 移动右边。

遍历整个数组，维持两个指针：left 和 right。当遍历到 0 是，将当前的 cur 和 left 互换。Left++;
如果遍历到 2，就将当前的 2 与 right 互换，right--;

77:1-N，挑选 K 个进行排列；

回溯+排列；

78：求数组的所有子集合。

回溯+组合；

80：删除有序数组中过于多余的数字，最多每个出现两次。

遍历数组，维持一个 left 指针，当前元素与 left-2 相等时，开始复制。否则继续走。

81：翻转部分的有序数组，可能存在重复，搜索 target;

二分查找的变形；

82：删除有序链表，将重复的全都删了；

Dummy，pre ,cur,将 cur 一直走一直走走到同一节点的最后面，如果 pre.next = cur;
就说明没有重复的，直接走，如果不是，是重复的，然后全都删了。

83：删除有序链表多余的节点，重复的保留一个。

如果当前和 pre 相等的话，直接将 cur 删除。否则继续双走。

86：链表，给定 target，将比 target 小的节点移动到左边。大的和等于的移动到后面。其他相对位置不变。

维持两个链表：一个比 target 小，其他的往第二个链表上挂。然后将第二个链表挂在第一个之后。

88：将两个有序数组合到第一个数组中；

两个数组之和为 len,将两个数组，两个指针，从尾到头，比较，大的放到 len—那。

89：题都暂时一眼看不懂

90：可能有重复元素的数组，求所有的组合。

回溯+组合。

91：将数字解码成 A-Z;1-26;,给出数字字符串，有几种解码方式：

Dp[n],以 index 为 n 结尾的子串，有几种编码方式，dp[0] 如果等于 0 就是 0，否则是 1；

然后倒数第一个在 0-9，dp[i]=dp[i-1];如果倒数俩在 10-26 之间，再加上个 dp[i-2]。

92：逆转链表的 m-n 这部分；

逆转链表基本试用头插法。将 cur 的 next 都放到 pre 之后。而逆转中间一部分。是指针先走两步，m--,n--;直到 m=1;然后头插逆转。逆转过程中，只逆转 n-m 个。

93：给定一数字字符串，输出可能的 IP 地址列表。

三重遍历，将字符串分割为(0，i)、(i、j)、(j,k)、(k,len)四部分，然后对每部分都进行合格性检查：既:长度小于 4，数小于 256，第一位等于 0 的话长度只能是 1;

94,二叉树中序遍历：

Stack,当 root 不为 null 或者栈非空时，root!=null 时。一路将左子树全部压入。然后弹出，走右子树。前序遍历，压入的时候就处理，中序，弹出的时候处理。后续，压入的时候处理，但先压右，然后压左。然后反转、

96：n 个节点，共有多少种二叉搜索树。

Dp[n] = dp[n-i]*dp[i-1];(1-n);

98:判断一个树，是否是二叉搜索树。

中序遍历，会得到一个有序数组。所以必然要当前节点的值大于前个节点。

```
if(pre != null && root.val <= pre.val) return false;
  pre = root;
```

100：判断两个树是否一致。

递归：如果都为空，一致；如果有一个为空或者值不一样，不一致。然后递归俩的左子树和右子树。

101：判断一个树是否对称。

将 is(root)转化成 is(root.left,root.right);递归调用：都为空，真。一空或值不同为假。然后 left.left 和 right.right 递归。

102：二叉树层序遍历。

加入队列，当队列不为空的时候：弹出时，先标记下队列的宽度。然后这是一层。弹出 n 个，并处理将子树加入。

103,之字形层序比那里

队列，整个 falg 正反标记位，正时候，list 正着加入，反时候，list 加入后 reverse.

104:二叉树的深度:

递归，或者层序 size 遍历。

105：前序和中序构成二叉树。

将中序加入 map(in[i],i);然后递归(pre,sp,ep,in,is,id,map);主要是要判断左子树有几个，右子树几个的位置。

106：中序和后序构成二叉树

一样；

107：层序遍历：从底层到上层；

一样的层序遍历：只不过每层都要直接放入队前。

108: 将有序数组转化成二叉搜索树

数组的中间(start+end)/2;这个作为根节点，然后左边的递归作为左子树，后边的递归作为右子树、

109：将有序链表转化成二叉搜索树：

理念和上面一样，链表的中间节点作为根节点，前面的作为左子树递归，后面的作为右子树递归。而找中间节点，就是 pre,fast,slow,fast 走两步，slow 走一步。Slow 就是根节点。但要将 pre 节点和后面的节点

断开，pre.next = null;

110:是否平衡二叉树：

Getdepth，在递归求深度的时候，要计算是否左右节点的高度是否大于 1；

111：叶节点到根节点的最短路径。

非递归的层序遍历：也是循环的是时候整个 size，但是如果弹出的时候，左子树且右子树都为 null，那证明叶节点来了，return 那个 level;

递归：

```
if (root == null)   return 0;
if (root.left == null)      return minDepth(root.right) + 1;
if (root.right == null) return minDepth(root.left) + 1;
return Math.min(minDepth(root.left),minDepth(root.right)) + 1;
```

112：二叉树是否存在从根节点到叶节点的路径和为 target:

因为这个比较简单，只需要判断是否有，而不是找出所有的路径。递归判断：root == null,不对。如果 root 没有左子树和右子树，且值等于 target,那直接返回 true。然后返回是否左右子树（left，target-val）；

113:找出所有路径和为 target 的序列

helper113(TreeNode root,int sum,List<List<Integer>> resList,List<Integer> tempList)

templist.add(root.val);如果是叶节点，且值为剩余的，就直接加入结果集。否则继续往下走。先左，后右，然后删除。

114：将二叉树往一边偏；

前序遍历？pre 和当前是左子树的关系；右子树都为空；

116：满二叉树，将为每个节点加上 next，就是每层的右边那个。

117：二叉树，将为每个节点加上 next，就是每层的右边那个。

125：字符串是否回文，只考虑数字和字母，大小写不论。

双指针，头尾，先将非字母和字符淘汰，然后转化为小写看看是否一致。不一样，直接 false：

129：二叉树，从头结点到尾节点：拼接成数字，然后所有的加一块。

递归求 sumRootToLeaf(root,sum);sum 为所有和拼接数字之和。

当 root null 直接返回 0!；当左子树和右子树都为 null 时，直接返回 sum*10+val;否则，左子树的递归加上右子树的递归。

136,一个数组，一个出现一次，其他的出现两次，求一次的那个

遍历：res = res ^ i;

137：数组，有人出现三次，有人出现一次，求一次的。

```
 int a=0,b=0;
for (int i : nums) {
      b = b^i & ~a;
      a = a^i & ~b;
}
return a|b;
```

138：带有 random 指针的复杂链表的复制：

HaspMap,每个都带值复制一下：放 key,value 里面。然后根据 key，value 的 next,random，这些都补上。

139：

举个例子：字符串为 applepen 和 wordDict 为 apple，pen。当我们遍历到字母 n 的时候我们从 n 开始向前找看看有没有 apple：即字符串被分为 app 和 lepen（因为 apple 的长度为 5，我们期待 lepen 可能为 apple，若 lepen 确实为 apple，app 也能够被 break（即 dp[2] = true)，那么我们将 dp[7] = true,但 lepen 不为 apple 而且 dp[2]为 false，因此我们循环到下一个 wordDict 即 pen，将字符串划分为前面的 apple 和后面的

pen，而后面的 pen 确实存在，dp[4] = true，所以确实可以划分，因此 dp[7]= true。

141. 链表是否有环：

快慢指针从头结点出发， 快的走两步，慢的走一步。当俩人走到一起时，就真的有环。

142：链表环入口节点

走到一起时，快从头开始，然后来人都齐步走，再次相遇时，为入口头结点。

143：重排链表：1-2-3-4-5；先第一，再最后一个，先第二个，再倒二，就这样。

这个先记住，过会再说，看剧呢，分心

144：前序遍历二叉树：

栈，先左子树，入栈，一直再左。然后弹出，当 cur = pop.rightl;入栈的时候打印。

146. LRU Cache

147：对链表进行插入排序，

148：NlongN 的时间排序链表。

150:逆波兰

151：单词逆序：

s.trim().split(" +");分割成数组：然后数组 Arrays.asList(strs),然后 Collections.reverse();

然后 String.join(“”,strs);

155:最小值的栈

一个栈正常走，另一个保存最小值。保持栈顶为最小，往下就变大。

160：两个链表的相交点

非递归就是，长度，然后长的走两步，然后齐步走。

165：比较版本号打大小，大于就返回 1，小于返回-1，相等返回 0；有些版本 1.01 是等于 1.001 的；

将字符串按.切割成数组。遍历到最大的数组长度。比较短的就当成 0，不为 0 的 Integer.parseInt();比较大小。

167：有序数组，找出两数之和为 target 的索引。

二分查找、

168：数字转化为 26 进制的 A-Z；

```
while(n != 0){
    n--;
    sb.insert(0, (char)('A' + n % 26));
    n /= 26;
}
```

169. Majority Element，超过一半的数字

摩尔投票；

171：26 进制 A-Z 转化为十进制数字

172：Given an integer n, return the number of trailing zeroes in n!.

```
while(n!=0){
    n = n/5;
    res+=n;
}
```

179：整数数组，求组合到一块的最大值。

将整数数组转化为字符串数组，然后 Arrays.sort(strs,new Comparator){};

然后 join;

187:DNA 序列，ACGT；求 10 长度的连续序列出现超过一次的序列。

字符串从第一个字母进行遍历，每次就截取 10 个字符。长度为 10 的字符串放入 set 中，如果放不进去，说明这个字符串以及有重复了，将这个放入结果集中。

189：数组左移 K 位

和字符串左移 K 为一样，更简单。翻转函数 Swap(int[] array,int s, int e);

先打翻转，然后 k%len,再进行两次翻转

199：二叉树，从右边看到的节点数组。也就是每层最右边那个。

还是层序遍历吧。每次循环开始的时候，先取 queue 的数量 K，然后弹出 K,最后一个 K 就是每层最右面那个。

200：dfs 求岛屿的个数。1 表示陆地，0 表示岛屿。给定一个矩阵，求岛屿的个数。岛屿的周围都是水，则是独立的一个。

因为陆地是相连的。我们遍历这个矩阵，第一次遇见 1 的时候，说明 count++;然后我们采用 dfs,将周围的 1 全部变成 2 或 0；这个 dfs 然后向四周扩散，当然边界也得出来，遇到不是 1 的就停止。这样，遇见一个 1 就把所有能连上的都变成 2，然后继续下一块岛屿。

201：N-M 数组的连续数，逐个&；求最后的结果。

也可以一个个遍历，从 M 干到 M;但也有规律。就是看着连续的数组，也就是 N 和 M 最右边有几位是相等的。而不等的，说明有 0 有 1；那就说明中间夹着的后面的位置都有 0；所以&的时候结果为 0；当 m!=n 的时候： m >>1 ,n >> 1; count++;然后再 m << count;

209:最短的连续子序列之和大于等于 target;

这玩意就得遍历； 维持两个指针，一个是当前走的指针，一个是计算开始的连续指针。还有一个是 sum 之和。I=j=0,sum=0;然后 i 先出发，然后 sum += nums[i];当和大于 target 时，min = Min(min.i-j+1);同时也让 j 开始走，并 sum-nums[j],看是否 j-i 这一点能否大于 target;

215：无序数组中找出最 K 大的数字：

最小堆 PriorityQueue(默认为最小堆)；

216：1-9 的数，选出 n 个，之和为 k。求所有 N 的组合。貌似一个数不能用两次

回溯+组合

220：一个数组，是否存在不同的数 i 和 j,使得 nums[i]-nums[j]的绝对值最大是 t,而 i-j 的绝对值最大是 k;

桶排序之类的，不太会。

221：矩阵，有 0 有 1；找出最大的正方形，其里面数值都 1；

```
dp[i][j] = Math.min(dp[i-1][j-1],Math.min(dp[i-1][j],dp[i][j-1]))+1;
result = Math.max(result,dp[i][j]);

    return result * result;
```

这里面 dp[i][j]是对角线和上面和左面那个，最小的 dp+1； 如果周围三个为 0，所以只能以他开始构建正方形，其他人已经不靠谱。如果其他人都为 1，那么他可以作为边长为 1 的正方形的右下角。如果周围都是 2，说明周围都有扩展，他可以为三。

222：给定一完全二叉树，求节点总个数。

要么直接遍历求个数。这个显然没有用到完全二叉树的概念。

要么对根节点，求最左的高度，最右的高度。如果他来一样。1<<leftDepth -1;如果不一样，递归求 1+count（左子树）+count(右子树)

223：以(A,B)和（C,D）为对角顶点，构建矩形，以 EF，GH 构建，求纵的矩形面积。

求 ABCD 面积和 EFGH 面积，求 AE，BF 的最大作为左底层，CG,DH 的最小作为上油层，如果左底小于上右，那么求这个重复面积，然后总减去这个得到总面积

227：非负整数，有+-*/和空格数字的字符串组成的表达式，求表达式的值；

整个栈。字符串从 0 开始遍历，如果是数字，继续进行，记得累加。如果遇到了非数字或者到头了，就要对上面的那个数字进行总结了，即现在遇到的 sign＝ “+-*/”;最开始时是+，所以：如果遇到+，就直接 push num,如果遇到-,就 push-num,如果遇到*，就将前面那个弹出，然后 push*num,如果遇到/就弹出然后 push 除去的结果。然后所有的弹出，然后求和；

228：给出排序好的数组，没有重复，将连续的整成 0->2 之类的形式，求所有的连续范围段。

也是数组从头遍历，如果下一个的值减去当今==1，那么证明这是个连续的，所以一直往下走，直到走到断开的地方。然后让当今的 cur 和 i 一直走的 nums[i]，看看是否相等，如果相等，说明这个人没跟人连续，直接加入结果集。如果不相等，就转化为 cur---nums[i]的方式,加入。

229，数组中超过 n/3 的次数的数。

摩尔投票变种：int countA = 0 ,countB = 0, res1 = 0 , res2 =0;对数组进行遍历，如果 res1，count1++；如果等于 res2,count2++;如果都不是，在 A=0 的情况下，对 A 进行转换成当前的。否则 B=0 的情况下，对 B 进行转换。如都不是，count1--,count2--;

然后再次遍历，看数组中 res1,res2 出现的次数是否超过了 n/3;

230:BST 二叉搜索树中，第 K 小的节点

中序遍历就是一个有序数组，遍历的时候记着，然后到 k 时，直接返回。2

236：二叉树中，两个节点最近的公共祖先节点。

递归的看的难受，看不懂

238：构建乘积数组

第一遍遍历，存前面的连城。第二遍遍历。存后面的连城，然后再相乘。

240：在矩阵中遍历，每行都增加，每列都增加。

从右上角开始遍历，如果小就下移，大就右移动。

241：数字和操作符，+-*,请随便加括号改变计算顺序，然后得出所有可能的结果。

这个有意思：这个就相当于分治；

我们首先遍历这个字符串。遇见加减乘号，将字符串分割成两部分，直到将所有的数字全部分成小段。

然后根据分段的符号，进行双层遍历，左边的取一个，右边的取一个，然后进行操作。结果放入 resList,进行返回。

260：数组，只有两个出线一次，其他的出现两次，求出现一次的那俩

首先遍历 diff ^= n； diff = diff & -diff; 然后再遍历，diff^n == 0 的一组，其他的一组。分别得出 res1,res2;

264：第 K 个丑数，因子只有 2,3,5 的数；

Dp[m] = min(dp[i]*2,dp[j]*3,dp[k]*5);维持 2,3,5 的指针，从 1 开始。一旦采用，就++；

274：if h of his/her N papers have at least h citations each, and the other N − h papers have no more than h citations each.

就是作者一共 N 个论文，其中 h 篇的引用都要大于等于 h，剩余的都不大于 h。这 h 就是他的引用数。

分成 N+1 个桶。0-N;将值大于等 N 的都放入 N 桶中，其他的都放在自己的序号中。

然后从 N 桶倒数 if(count >= i)　　return i;

这个 count 的计算方式:将大于等于 H 的都统计了，剩余的都在小 h 桶，也符合规矩。

275：同上，也是求 H。但这个数组是有序的；

有序的，就整二分查找。主要是看 citations[mid]<index 的关系，index 就是右半边的大数有几个。

287：N+1 数，都是 1-n 之间，求重复的那个数。

```
int slow = 0,fast = 0;
while (true){
```

```
            slow = nums[slow];
            fast = nums[nums[fast]];
            if(slow == fast)
                break;
        }
        fast = 0;
        while (fast != slow){
            fast = nums[fast];
            slow = nums[slow];
        }
        return slow;
```

300：最长递增序列，可以不连续。但是前面的 index 比后面的小。

求 dp[n]:当然，整个遍历。然后再遍历比当前 index 小的， 当 nums[i]>nums[j]时，dp[i] = Max（dp[i],dp[j]+1）;然后求 dp[n]中最大的。

306：数字字符串，我们将字符串分割最少三个数字，然后 Fn = Fn-1+Fn-2;

一下子看不懂了。

310：一个图，给的形式是 n 个节点，然后节点间关系相连的[][].以一个节点为根节点，求得到的最小高度的树的根节点。

313：超级丑数，求第 N 个丑数，并且给出数组因子，只有只包括该数组因子的数才是超级丑数

Dp[i] = max(dp[i],dp[index[j]]*pre[j]);

318:字符串数组，求两个没有公共字母的字符串长度乘积的最大值；

看不懂，

319：开关转换，刚开始有 N 的灯，最初都是灭的，第一轮，每一个都按一个。第二轮，每二个按一个开关。第 N 轮之后，亮着的灯的个数。

```
return (int)Math.sqrt(n);
```

322：给一堆零钱，可以重复使用，然后给整钱。将整钱换成零钱，求最小的零钱数，如果不能换，返回-1；

324：无序数组重排序：nums[0] < nums[1] > nums[2] < nums[3]....

他这是，重新复制一个数组，然后排序。然后遍历原数组，将排序好的前一半放到偶数位，后一半放到奇数位。

328：链表重排序，将奇数位 index,而不是奇数值的放到前面。偶数 index 的放后面：

跳着来，将 odd.next = odd.next.next;even.next = even.next.next;这样就将 odd 的串一块，even 的串一块。然后将 old 指向奇数的开头。

331：将一个二叉树，他的空子节点记成#，给定一个字符串，问这个是不是前序遍历；

332:航班信息[from,to],一些列的这种数组信息，这人从 JFK 出发，求他的真正路线一次经过的城市；

334：数字数组，是否存在三个子序列，可以不连续，是递增的；

Return true if there exists i, j, k

such that arr[i] < arr[j] < arr[k] given 0 ≤ i < j < k ≤ n-1 else return false.

```
    int m1=Integer.MAX_VALUE,m2=m1;
        for(int num:nums){
            if(num <= m1)
                m1 = num;
            else if(num <= m2)
                m2 = num;
```

```
            else
                return true;
        }
        return false;
```

338：给定一 num.求 0-num 这些数每个的二进制中 1 的个数。

res[i] = res[i>>1]+(i&1);

347，TopK 出现频率最高的数字，一个数组，每个数字可能出现多次，求最 K 多出现的那 k 个

先用 hashmap 统计每个数字出现的次数，key 是数字，value 是出现的次数。

然后弄个优先队列，

343：给定一个整数，将他分成最小二段，和为这个整数，求这些段乘积的最大值。

```
dp[i] = Math.max(dp[i],Math.max(j, dp[j]) * Math.max(i - j, dp[i - j]));
```

357:n 位数的数，求没有重复数字的总个数。11,121 这种就不行。

```
if (n==0)
    return 1;
int sum    =10;
int q = 9;
for (int i =1; i<n; i++){
    q *= (10-i);
    sum += q;
}
return sum;
```

头条：

1、数组的逆序数　//归并排序时，都从右到左比那里，比较大小时，如果左边大，逆序数加；

2、LRU　//hashMap 加双向链表，双向链表有头尾节点，key,value 封装成 Node,hashMap 的 key 为 key,value 为 node 节点。node 节点又在双向链表中

3、最长回文序列　leetocde 5 ：　从左到右遍历时，并扩展，看能扩到哪。记录最大和最左

4、矩阵中的最长递增路径，可以上下左右一起都走；　leetcode329　//双层遍历进行 dfs,维持一个二维矩阵。如果矩阵不为 0，max=1,往四个方向遍历，如果符合条件，进行扩展。len = 1+ dfs();然后求 max,最后将 max 赋值给数组。

5、判断一个二叉树是另一个二叉树的子树 剑指　//2 是否为空，1 是否为空或者是否相等，左子树和右子

6、归并排序的时间复杂度　// NlongN

7、求给出 01 矩阵中的最大正方形面积（全为 1）lc221　dp[i-1][j-1] = min{dp[i-1][j-1],dp[i][j-1],dp[i-1][j]}+1;

8、求二叉树中距离最远的节点　leetcode543　// 递归求根的深度，max = Math.max(max,left+right);

9、判断字符串是否为合法 IPV4 地址　　lc468//对.切割，是否长度为 4，且每个大于 0 小于 256

10、数组值为 1-n，各出现一次，先加入 x（x 也是 1-n 的范围），找出 x //将 A[i]放到 i 位置，如果不是，while(A[i] !=　i) if(A[i] = A[A[i]) 这个重复，否则，i 和 A[i]位置的交换

11、给定 n，计算 15n，不用+*/　　n<<4 – 15;

12、给定字符数组 chars，将其右移 n 位 先将整个数组翻转，然后将 len-n 翻转，n 翻转

13、100 层楼，只有两个鸡蛋，找出鸡蛋会在哪一层楼被摔碎//x+x-1+x-2+….1 = 100;

14、reverse linked list in a group of k //先求长度，然后 for(int j = len;j>=k;j = j-k) 然后遍历一段时间，头插。

15、如何空间 O(0)实现两个数的互换 a = a +b; b = a-b; a = a-b;

16、IP 地址的 Regex // 25[0-5]|2[0-4]/d{1} | [0-1]?\d{0,2}

17、the longest path in a binary tree　lc124

```
if(root == null) return 0;
int left = Math.max(0,MaxPathDown(root.left));
int right = Math.max(0,MaxPathDown(root.right));
maxValue = Math.max(maxValue,left+right+root.val);
return Math.max(left,right)+root.val;
```

18、the largest consecutive sum in an array dp[i] = max(dp[i-1]+ nums[i],nums[i]);

19、LeetCode 41 Find missing positive 将符合条件的，即大于 0 小于 n 的，进行转换。换到该去的地方 A[i]放到 i-1 地方。

20、给一个小于一亿的中文数字字符串,转化成数字格式

21、一个数组,把所有的 0 都移动到末尾,还要保持顺序不乱 维持临界点 j,如果当前遍历不是 0，就和 j 互换

22、罗马数字转整数 leetcode13 //罗马数字特点，如果这个比后面的小，减去，否则加上。

23、二叉树的序列化和反序列化//层序遍历

24、输入一个数组，输出数组中满足条件的数字，条件为：数组中当前元素的值大于等于它前面所有的元素，小于等于它后面所有的元素。// 维持一个数组，以及长度，以及目前的 max.如果当前大于 max，入队。否则，将队里面的慢慢退。

25、给出一个数字，对数字的两位进行交换，只能交换一次，输出可能结果中的最小数字//桶排序，table＝new int[10];算出每个数字最后出现的位置。从前面遍历，从 9 开始遍历，如果 table[i]存在，就交换。

26、输入一个字符串，字符串中字符全部为数字，在字符串中插入 '.' 使得结果为合法的 ip 地址，输出全部可能的结果

27、基数排序

28、是否为回文结构。一快一慢，走到中间，然后逆转链表。然后走，看是否回文

29、最大不重复子串 set 以及前面的 j;往前遍历，如果 set 不重复，加入，否则将 j++的一个个弹出去。

30、复杂链表复制//

31、长度为 n 的数组，元素大小是 0~n-1,判断数组元素是否有重复的

32、list1/list2 交替打印元素

33、36 进制加法

34、合并区间

35、快排

36、生产者-消费者 模型

37、排序一个字符串时间要求 O(n)

38、给一个有重复数字的数组，求集合{(a,b,c) | a+b+c=0}

39、两个栈实现队列

40、二叉树转化为双端链表

41、手写线程池

42、之字形打印二叉树

43、给定一个数组，调整该数组，使其满足堆的性质

44、给定 n 个单词，如果单词组成一致但是元素顺序不一致，该对单词为同位词，例如：abc,bca 为同位词．求所有同位词的集合输出

45、链表，两个链表的公共点

46、二叉树的后续遍历非递归形式

47、买卖股票的最佳时机，只能一次买入和一次卖出

48、可以进行多次交易的结果，求赚取的最大利润

49、(A,B)(A,C)(B,D)(D,A)判断是否有循环引用，提示用拓扑排序

50、数组找是否存在和为 M 的两个数

51、KMP
52、实现一个阻塞队列（生产者消费者模型）
53、找出 10000 个数据中第 k 大的数
54、输入一个字符串，包含数字、加减乘除和括号，输出结果，编程
55、给定一个数字 x，要求使用 k 个数字求和可以得到 x，数字从 1-9 中选择，不能重复。
56、输入一个正整数 N，返回 N 个 '(' 和 N 个 ')' 的所有可能情况
57、76.minimum-window-substring、30.substring-with-concatenation-of-all-words、4.trapping-rain-water，
58、求树的最左下节点
59、无序数组中第 k 大的数（quick select）
60、求旋转数组找最小值（二分）
61、判断二叉树是否镜像（递归）
62、给定一个矩阵，从左上角开始只能往下或者右走，求到达右下角的最小权值路径
63、字符串转 Int，如果越界就返回 0
64、lc400
65、单向链表实现加法
66、打家劫舍
67、收到礼物最大值
68、五张牌，其中大小鬼为癞子，牌面为 0，判断这五张牌是否能组成顺子
69、给定一个字符串打印所有的子串，要求不重复
70、自然数 1-n,排一块组成的字符串，求第 k 位是什么，12345678910，如何第 10 位是 0；