面试题1：二维数组中的查找

题目：在一个二维数组中，每一行都按照从左到右递增的顺序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样一个二维数组和一个整数，判断数组中是否含有该整数。

```
public static boolean find(int[][] array, int rows, int columns, int number){
        boolean found = false;
        int row = 0;
        int column = columns -1;
        while(row < rows && column >= 0){
            if(array[row][column] == number){
                found = true;
                break;
            }else if(number > array[row][column]){
                row++;
            }else if(number < array[row][column]){
                column--;
            }
        }
        return found;
    }

    public static void main(String[] args) {
        int[][] array = {{1, 2, 8, 9},{2, 4, 9, 12},{4, 7, 10, 13},{6, 8, 11, 15}};
        System.out.println(find(array, 4, 4, 1111));
    }
```

面试题2.替换空格

题目：请实现一个函数，把字符串中的每个空格替换成“%20”。例如输入“We are happy.”，则输出“We%20are%20happy.”

```
public static void replaceBlank(char[] str, int length){
        if(str == null || str.length <= 0)return;
        int originLength = 0; //原数组长度
        int numberOfBlank = 0; //空格数量
```

```
        int i = 0;//指针1
        while(str[i] != '\0'){
            if(str[i] == ' '){
                numberOfBlank++;
            }
            originLength++;
            i++;
        }
        int newLength = originLength + numberOfBlank*2;
        if(newLength > length)return;
        int j = newLength;//指针2
        while(i != j){
            if(str[i] == ' '){
                str[j--] = '0';
                str[j--] = '2';
                str[j--] = '%';
                i--;
            }else{
                str[j--] = str[i--];
            }
        }
    }
    public static void main(String[] args) {
        char[] str = new char[]{'a','r',' ',' ','h','a','p','p','y','.','\0'};
        char[] arr = Arrays.copyOf(str, 30);
        System.out.println(Arrays.toString(arr));
        replaceBlank(arr, arr.length);
        System.out.println(Arrays.toString(arr));
    }
```

面试题3：从尾到头打印链表

题目：输入一个链表的头结点，从尾到头反过来打印每个结点的值。

```
    //递归倒序打印链表
    public static void printListRecursively(Node head){
```

```
        if(head != null){
            if(head.next != null){
                printListRecursively(head.next);
            }
            System.out.println(head.key);
        }
    }

    //使用栈倒序打印链表
    public static void printListByStack(Node head){
        Stack<Node> stack = new Stack<Node>();
        Node tmp = head;
        while(tmp != null){
            stack.push(tmp);
            tmp = tmp.next;
        }
        while(true){
            if(!stack.isEmpty()){
                Node node = stack.pop();
                System.out.println(node.key);
            }
        }
    }
```

面试题4：用两个栈实现队列

题目：用两个栈实现一个队列。队列的声明如下，请实现它的两个函数appendTail和deleteHead,分别完成在队列尾部插入结点和在队列头部删除结点的功能。

```
public class MyQueue<T> {
    private Stack<T> a = new Stack<T>();
    private Stack<T> b = new Stack<T>();
    public T deleteHead(){
        if(b.isEmpty()){
            if(a.isEmpty()){
```

```
                throw new RuntimeException("没有更多的元素可以获取！");
            }else{
                while(!a.isEmpty()){
                    b.push(a.pop());
                }
                return b.pop();
            }
        }else{
            return b.pop();
        }
    }
    public void appendTail(T item){
        if(item == null)return;
        a.push(item);
    }
}
```

面试题5：旋转数组的最小数字

题目：把一个数组最开始的若干元素搬到数组的末尾，我们称之为数组的旋转。输入一个递增排序的数组的一个旋转，输出旋转数组的最小元素。例如数组{3,4,5,1,2}为{1,2,3,4,5}的一个旋转，该数组的最小值为1。

```
    public static int min(int[] array){
        if(array == null || array.length <= 0){
            throw new RuntimeException("参数异常！");
        }
        int index1 = 0;
        int index2 = array.length - 1;
        int mid = index1;
        while(array[index1] >= array[index2]){
            if((index2 - index1) == 1){
                return array[index2];
            }
            mid = (index1 + index2)/2;
            //处理特殊情况，例如1,0,1,1,1
```

```
                if(array[index1] == array[mid] && array[index2] ==
array[mid]){
                    return midInOrder(array, index1, index2);
                }
                if(array[mid] >= array[index1]){
                    index1 = mid;
                }else if(array[mid] <= array[index2]){
                    index2 = mid;
                }
            }
            return array[mid];
        }

        public static int midInOrder(int[] array, int index1, int index2){
            int result = array[index1];
            for(int i = index1 + 1; i <= index2; i++){
                if(array[i] < result){
                    result = array[i];
                }
            }
            return result;
        }
```

面试题6：斐波那契数列

题目一：写一个函数，输入n,求斐波那契（Fibonacci）数列的第n项。斐波那契数列的定义如下：

$$f(n)=\begin{cases}0 & n=0\\ 1 & n=1\\ f(n-1)+f(n-2) & n>1\end{cases}$$

```
        public static long fibonacci(int n){
            int[] result = {0, 1};
            if(n < 2)return result[n];
            int a = 0;
```

```
        int b = 1;
        int tmp = 0;
        for(int i = 2; i <= n; i++){
            tmp = a + b;
            a = b;
            b = tmp;
        }
        return tmp;
    }
```

题目二：一只青蛙一次可以跳上1级台阶，也可以跳上两级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

面试题7：二进制中1的个数

题目：请实现一个函数，输入一个整数，输出该数的二进制表示1的个数。例如把9表示成二进制是1001，有2个1。因此如果输入9，该函数输出2。

```
    public static int numberOf1(int n){
        int count = 0;
        while(n != 0){
            ++count;
            n=(n-1)&n;
        }
        return count;
    }
```

总结：把一个整数减去1之后再和原来的整数位做位与运算，得到的结果相当于是把整数的二进制表示中的最后一个1变成0。很多二进制的问题都可以用这个思路解决。

面试题8：数值的整数次方

题目：实现函数double Power(double base, int exponent),求base的exponent次方。不得使用库函数，同时不需要考虑大数问题。

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & n\text{为偶数} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & n\text{为奇数} \end{cases}$$

```
    public static double powerWithUnsignedExponent(double base, int exponent){
```

```
            if(exponent == 0)return 1;
            if(exponent == 1)return base;
            double result = powerWithUnsignedExponent(base, exponent
>> 1);
            result *= result;
            if((exponent & 0x1) == 1){
                result *= base;
            }
            return result;
        }
```

总结：用右移运算符代替了除以2，用位与运算符代替了求余运算符（%）来判断一个数是奇数还是偶数。

面试题9：利用递归实现字符串反转

```
    public static String toReverseString(String s){
        if(s == null || "".equals(s.trim()))return s;
        return reverseString(s, 0, s.length()-1);
    }

    public static String reverseString(String s, int i, int j){
        char[] a = s.toCharArray();
        if(i < j){
            char tmp = a[i];
            a[i] = a[j];
            a[j] = tmp;
            s = reverseString(String.valueOf(a), ++i, --j);
        }
        return s;
    }
```

或者

```
    public static String reverseString(String s){
        if(s == null || "".equals(s.trim()) || s.length() <= 1)return s;
        String left = s.substring(0, s.length() >> 1);
        String right = s.substring(s.length() >> 1, s.length());
```

```
            return reverseString(right) + reverseString(left);
        }
```

总结：很多时候递归的思想里面包含了分治算法的思想在里面。

面试题10：打印1到最大的n位数

题目：输入数字n,按顺序打印从1最大的n位十进制数。比如输入3，则打印1、2、3一直到最大的3位数即999。

```
        public static boolean increment(char[] number){
            boolean isOverflow = false;
            int nTakeOver = 0;
            for(int i = number.length-1; i >= 0; i--){
                int nSum = number[i] - '0' + nTakeOver;
                if(i == number.length-1)nSum++;
                if(nSum >= 10){
                    if(i == 0){
                        isOverflow = true;
                    }else{
                        nSum -= 10;
                        nTakeOver = 1;
                        number[i] = (char) ('0' + nSum);
                    }
                }else{
                    number[i] = (char) ('0' + nSum);
                    break;
                }
            }
            return isOverflow;
        }

        public static void printToMaxOfDigits(int n){
            if(n <= 0)return;
            char[] number = new char[n];
            Arrays.fill(number, '0');
            while(!increment(number)){
```

```
                System.out.println(String.valueOf(number));
            }
        }
```

或者

```
        public static void print1ToMaxOfNDigitsRecursively(char[] number,
int index){
            if(index == number.length){
                System.out.println(String.valueOf(number));
                return;
            }
            for(int i = 0; i < 10; i++){
                number[index] = (char)('0' + i);
                print1ToMaxOfNDigitsRecursively(number, index + 1);
            }
        }

        public static void print1ToMaxOfNDigits(int n){
            if(n <= 0)return;
            char[] number = new char[n];
            Arrays.fill(number, '0');
            print1ToMaxOfNDigitsRecursively(number, 0);
        }
```

总结：考察解决大数的问题的能力

面试题13：在O(1)时间删除链表结点（只遍历一次链表）

题目：给定单向链表的头指针和一个结点指针，定义一个函数在O(1)时间删除该结点。

```
        public void deleteNode(Node pListHead, Node node){
            if(pListHead == null || node == null)return;
            if(node.next != null){//普通删除一个结点
                Node tmp = node.next;
                node.value = node.next.value;
                node.next = node.next.next;
                tmp.next = null;
```

```
        }else if(node == pListHead){//链表只有一个结点
            pListHead = null;
        }else{//待删除的结点为尾结点
            Node tmp = first;
            while(tmp.next != node){
                tmp = tmp.next;
            }
            tmp.next = null;
        }
    }
```

总结：1.当我们想要删除一个结点时，并不一定要删除这个结点本身。可以先把下一个结点的内容复制出来覆盖被删除节点的内容，然后把下一个结点删除。2.考虑删除结点位于链表的尾部及输入的链表只有一个结点这些特殊情况。

面试题14：调整数组顺序使奇数位于偶数前面

题目：输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有奇数位于数组的前半部分，所有偶数位于数组的后半部分

```
    public static void ReorderOddEven(int[] array){
        if(array == null || array.length == 0)return;
        int i = 0;
        int j = array.length - 1;
        while(i < j){
            while((i < j) && isEven(array[i]))++i;
            while((i < j) && !isEven(array[j]))--j;
            int tmp = array[i];
            array[i] = array[j];
            array[j] = tmp;
        }
    }

    public static boolean isEven(int n){
        return (n & 0x1) == 1;
    }
```

面试题15：链表中倒数第K个结点

题目：输入一个链表，输出该链表中倒数第K个结点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾结点是倒数第1个结点。例如一个链表有6个结点，从头结点开始他们的值依次为1、2、3、4、5、6。这个链表的倒数第三个结点的值为4的结点。

```
public Node findKthToTail(Node pListHead, int k){
    if(pListHead == null || k == 0)return null;
    //定义两个指针pAhead、pBehind，其中pAhead先走k步
    Node pAhead = pListHead;
    Node pBehind = pListHead;
    for(int i = 0; i < k-1; ++i){
        if(pAhead.next != null){
            pAhead = pAhead.next;
        }else{
            return null;
        }
    }
    while(pAhead.next != null){
        pAhead = pAhead.next;
        pBehind = pBehind.next;
    }
    return pBehind;
}
```

总结：当我们用一个指针遍历链表不能解决问题的时候，可以尝试用两个指针来遍历链表。可以让其中一个指针遍历的速度快些（比如一次在链表上走两步），或者让它在链表上走若干步。

面试题16：反转链表

题目：定义一个函数，输入一个链表的头结点，反转该链表并输出反转后链表的头结点。链表结点定义如下：

```
public Node reverseList(Node pHead){
    if(pHead == null)return null;
    if(pHead.next == null)return pHead;
    Node pNode = pHead;
```

```
        Node pPrev = null;
        while(true){
            Node tmp = pNode.next;
            pNode.next = pPrev;
            if(tmp == null)break;
            pPrev = pNode;
            pNode = tmp;
        }
        return pNode;
    }
```

总结：1.需要将当前结点的next指针由存放后下一个结点的值改为存放上一个节点的值。

2.考查代码的鲁棒性，对一些特殊情况的处理

面试题17：合并两个排序的链表

题目：输入两个递增排序的链表，合并这两个链表并使链表中的结点仍然是按照递增排序的。例如输入图3.7中的链表1和链表2，则合并之后的升序链表如链表3所示。链表结点定义如下：

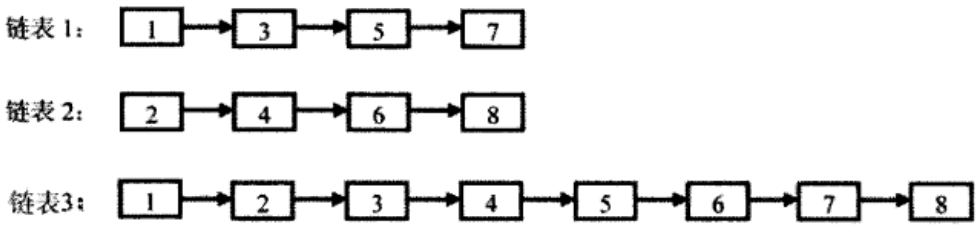

图 3.7　合并两个排序链表的过程

注：链表 1 和链表 2 是两个递增排序的链表，合并这两个链表得到升序链表为链表 3。

```
public Node merge(Node pHead1, Node pHead2){
    if(pHead1 == null){
        return pHead2;
    }else if(pHead2 == null){
        return pHead1;
    }
    Node pMergeHead = null;
    if(pHead1.value < pHead2.value){
```

```
            pMergeHead = pHead1;
            pMergeHead.next = merge(pHead1.next, pHead2);
        }else{
            pMergeHead = pHead2;
            pMergeHead.next = merge(pHead2.next, pHead1);
        }
        return pMergeHead;
    }
```

总结：递归+分治思想

面试题18：已知有两个有序数组，请编写一个函数，实现两个数组合并，并且合并后的数组也是有序的。

```
    public int[] merge(int[] arr1, int[] arr2){
        if(arr1 == null || arr1.length == 0 || arr2 == null || arr2.length == 0)return null;
        int[] output = new int[arr1.length + arr2.length];
        int i = 0, j = 0, k = 0;
        while(i < arr1.length && j < arr2.length){
            if(arr1[i] <= arr2[j]){
                output[k++] = arr1[i++];
            }else{
                output[k++] = arr2[j++];
            }
        }
        while(i < arr1.length){
            output[k++] = arr1[i++];
        }
        while(j < arr2.length){
            output[k++] = arr2[j++];
        }
        return output;
    };
```

总结：三个指针分别指向两个需要合并的数组和结果输出数组。

面试题18：树的子结构

题目：输入两棵二叉树A和B,判断B是不是A的子结构。二叉树结点的定义如下：

```
    /**
     * 判断是否有相同的子树
     * @param node1
     * @param node2
     * @return
     */
    public boolean hasSubtree(Node node1, Node node2){
        if(node1 == null || node2 == null)return false;
        boolean result = false;
        if(node1.value == node2.value){
            result = doesTree1HaveTree2(node1, node2);
        }
        if(!result)result = hasSubtree(node1.left, node2);
        if(!result)result = hasSubtree(node1.right, node2);
        return result;
    }

    /**
     * 判断两个子树是否完全相同
     * @param node1
     * @param node2
     * @return
     */
    public boolean doesTree1HaveTree2(Node node1, Node node2){
        if(node2 == null)return true;
        if(node1 == null)return false;
        if(node1.value != node1.value)return false;
        return doesTree1HaveTree2(node1.left, node2.left) &&
doesTree1HaveTree2(node1.right, node2.right);
    }
```

面试题19：二叉树的镜像（反转二叉树）

题目：请完成一个函数，输入一个二叉树，该函数输出它的镜像

```
public void mirrorRecursively(Node phead){
    if(phead == null || (phead.left == null && phead.right == null))return;
    Node tmp = phead.left;
    phead.left = phead.right;
    phead.right = tmp;
    if(phead.left != null)mirrorRecursively(phead.left);
    if(phead.right != null)mirrorRecursively(phead.right);
}

public class Node{
    public Node left;
    public Node right;
    public int value;
}
```

面试题21：包含min函数的栈

题目：定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的min函数。在该栈中，调用min、push及pop的时间复杂度都是O(1)。

```
public Stack<Integer> stack1 = new Stack<Integer>();
public Stack<Integer> stack2 = new Stack<Integer>();

public int min(){
    return stack2.peek();
}

public void push(int n){
    if(stack1.isEmpty() && stack2.isEmpty()){
        stack1.push(n);
        stack2.push(n);
    }else{
        if(n < stack2.peek()){
            stack1.push(n);
            stack2.push(n);
```

```
            }else{
                stack1.push(n);
                stack2.push(stack2.peek());
            }
        }
    }

    public int pop(){
        stack2.pop();
        return stack1.pop();
    }
```

面试题22：栈的压入、弹出序列

题目：输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假如压入栈的所有数字均不相等。例如序列1、2、3、4、5是某栈的压栈序列，序列4、5、3、2、1是该压栈序列对应的一个弹出序列，但4、3、5、1、2就不可能是该压栈序列的弹出序列。

```
    public boolean isPopOrder(int[] arr1, int[] arr2){
        if(arr1 == null || arr2 == null || arr1.length != arr2.length)return false;
        boolean result = true;
        int i = 0;
        int j = 0;
        Stack<Integer> stack = new Stack<Integer>();
        while(i < arr1.length && j < arr2.length){
            if(arr1[i] != arr2[j]){
                stack.push(arr1[i++]);
            }else{
                i++;
                j++;
            }
        }
        while(!stack.isEmpty() && j < arr2.length){
            if(stack.pop() != arr2[j++]){
                result = false;
```

```
                    break;
                }
            }
            return result;
        }
```

面试题23：从上往下打印二叉树

题目：从上往下打印二叉树的每个结点，同一层的结点按照从左到右的顺序打印。例如输入图4.5中的二叉树，则依次打印出8、6、10、5、7、9、11。

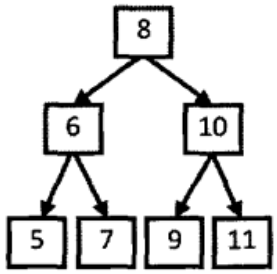

图 4.5　一棵二叉树，从上往下按层打印的顺序为 8、6、10、5、7、9、11

```
    public void printFromTopToBottom(Node node){
        if(node == null)return;
        ConcurrentLinkedQueue<Node> queue = new
ConcurrentLinkedQueue<Node>();
        queue.add(node);
        while(!queue.isEmpty()){
            Node currentNode = queue.poll();
            System.out.print(currentNode.value + " ");
            if(currentNode.left != null)queue.add(currentNode.left);
            if(currentNode.right != null)queue.add(currentNode.right);
        }
    }
```

补充：

面试题：按层遍历二叉树，打印最右边的结点

题目：输出二叉树每层最右端节点的数值

```
    public void printRightNode(Node head){
```

```
        if(head == null)return;
        ConcurrentLinkedQueue<Node> a = new
ConcurrentLinkedQueue<Node>();
        ConcurrentLinkedQueue<Node> b = new
ConcurrentLinkedQueue<Node>();
        a.add(head);
        while(!(a.isEmpty() && b.isEmpty())){
            while(!a.isEmpty()){
                Node tmp = a.poll();
                if(tmp.left != null)b.add(tmp.left);
                if(tmp.right != null)b.add(tmp.right);
                if(a.isEmpty())System.out.println(tmp.value);
            }
            while(!b.isEmpty()){
                Node tmp = b.poll();
                if(tmp.left != null)a.add(tmp.left);
                if(tmp.right != null)a.add(tmp.right);
                if(b.isEmpty())System.out.println(tmp.value);
            }
        }
    }
```

面试题24：二叉搜索树的后序遍历序列

题目：输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历的结果。如果是则返回true,否则返回false。假设输入的数组的任意两个数字都互不相同。

```
    /**
     * 判断序列是否是二叉树后序遍历结果
     * @param arr
     * @param low
     * @param high
     * @return
     */
    public boolean verifySquenceOfBTS(int[] arr, int low, int high){
        if(low == high)return true;
```

```
        //定义指针i从low遍历到high寻找到切分点j
        int i = low;
        while(i < high){
            if(arr[i++] > arr[high])break;
        }
        //寻找切分点
        int j = i <= high ? i-1 : low;
        while(i < high){
            if(arr[i++] < arr[high])return false;
        }
        boolean left = true;
        boolean right = true;
        if(j <= (i-1)){
            left = verifySquenceOfBTS(arr, low, j);
        }
        if(j < high){
            right = verifySquenceOfBTS(arr, j+1, high);
        }
        return left && right;
    }

    /**
     * 预处理边界问题
     * @param arr
     * @return
     */
    public boolean prepareVerifySquenceOfBTS(int[] arr){
        if(arr == null || arr.length == 0)return false;
        return verifySquenceOfBTS(arr, 0, arr.length -1);
    }
```

面试题29：数组中出现次数超过一半的数字

题目：数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。例如输入一个长度为9的数组{1,2,3,2,2,2,5,4,2}。由于数字2在数组中出现了5次，超过数组长度的一半，因此输出2。

```
public int moreThanHalfNum(int[] number){
    if(number == null || number.length == 0)return 0;
    int low = 0;
    int high = number.length - 1;
    int middle = (number.length - 1) >> 1;
    int index = partition(number, low, high);
    while(index != middle){
        if(index > middle){
            index = partition(number, low, index-1);
        }else{
            index = partition(number, index+1, high);
        }
    }
    int result = number[index];
    if(!checkMoreThanHalf(number, result)){
        result = 0;
    }
    return result;
}

public int partition(int[] array, int low, int high){
    int tmp = array[low];
    int i = low;
    int j = high;
    while(i<j){
        while(i < j && array[j] >= tmp){
            j--;
        }
        if(i < j){
            array[i++] = array[j];
        }
```

```
            while(i < j && array[i] <= tmp){
                i++;
            }
            if(i < j){
                array[j--] = array[i];
            }
        }
        array[i] = tmp;
        return i;
    }

    public boolean checkMoreThanHalf(int[] array, int number){
        boolean isMoreThanHalf = true;
        int count = 0;
        for(int i = 0; i < array.length; i++){
            if(array[i] == number){
                count++;
            }
        }
        if((count << 1) <= array.length){
            isMoreThanHalf = false;
        }
        return isMoreThanHalf;
    }
```

或者

```
    public int moreThanHalfNum(int[] number){
        if(number == null || number.length == 0)return 0;
        int result = number[0];
        int times = 1;
        for(int i = 1; i < number.length; i++){
            if(times == 0){
                times = 1;
                result = number[i];
            }else if(number[i] == result){
```

```
                times++;
            }else{
                times--;
            }
        }
        if(!checkMoreThanHalf(number, result)){
            result = 0;
        }
        return result;
    }

    public boolean checkMoreThanHalf(int[] array, int number){
        boolean isMoreThanHalf = true;
        int count = 0;
        for(int i = 0; i < array.length; i++){
            if(array[i] == number){
                count++;
            }
        }
        if((count << 1) <= array.length){
            isMoreThanHalf = false;
        }
        return isMoreThanHalf;
    }
```

面试题30：最小的K个数

题目：输入n个整数，找出其中最小的k个数。例如输入4、5、1、6、2、7、3、8这8个数字，则最小的4个数字是1、2、3、4。

表 5.1　两种算法的特点比较

	基于Partition函数的思路	基于堆或者红黑树的思路
时间复杂度	O(n)	O(n*logk)
是否需要修改输入数组	是	否
是否适用于海量数据	否	是

```
    public void getLeastNumbers(int[] input, int k, int[] output){
        if(input == null || output == null || input.length == 0 || output.length
== 0 ||
         output.length != k || input.length < k)return;
        int low = 0;
        int high = input.length - 1;
        int index = partition(input, low, high);
        while(index != (k-1)){
            if(index > (k-1)){
                index = index - 1;
                index = partition(input, low, index);
            }else{
                index = index + 1;
                index = partition(input, index, high);
            }
        }
        for(int i = 0; i < k; i++){
            output[i] = input[i];
        }
    }

    public int partition(int[] array, int low, int high){
        int tmp = array[low];
        int i = low;
        int j = high;
        while(i < j){
            while(i < j && array[j] >= tmp){
                j--;
            }
            if(i < j){
                array[i++] = array[j];
            }
            while(i < j && array[i] <= tmp){
                i++;
```

```
                }
                if(i < j){
                        array[j--] = array[i];
                }
        }
        array[i] = tmp;
        return i;
    }
```

或者

面试题31：连续子数组的最大和

题目：输入一个整型数组，数组里有正数也有负数。数组中一个或连续的多个整数组成一个子数组。求所有子数组的和的最大值。要求时间复杂度为O(n)

```
    public static int findGreatestSumOfSubArray(int[] array){
        if(array == null || array.length == 0)return 0;
        int nCurSum = 0;
        int nGreatestSum = Integer.MAX_VALUE;
        for(int i = 0; i < array.length; ++i){
            if(nCurSum <= 0){
                nCurSum = array[i];
            }else{
                nCurSum += array[i];
            }
            if(nCurSum > nGreatestSum){
                nCurSum = nGreatestSum;
            }
        }
        return nCurSum;
    }
```

注意处理大数问题

面试题34：丑数

题目：我们把只含有因子2、3和5的数称为丑数（Ugly Number）。求按从小到大

的顺序的第1500个丑数。例如6、8都是丑数，但14不是，因为它包含因子7。习惯上我们把1当做第一个丑数。、

```
    public int getUglyNumber(int n){
        if(n <= 0)return 0;
        int[] uglyNumbers = new int[n];
        uglyNumbers[0] = 1;
        int a = 0;
        int b = 0;
        int c = 0;
        int index = 1;
        while(index < n){
            uglyNumbers[index] = min(uglyNumbers[a]*2,
uglyNumbers[b]*3, uglyNumbers[c]*5);
            if(uglyNumbers[a]*2 <= uglyNumbers[index]){
                a++;
            }
            if(uglyNumbers[b]*3 <= uglyNumbers[index]){
                b++;
            }
            if(uglyNumbers[c]*5 <= uglyNumbers[index]){
                c++;
            }
            index++;
        }
        return uglyNumbers[n-1];
    }

    public int min(int num1, int num2, int num3){
        int min = num1 < num2 ? num1 : num2;
        return min < num3 ? min : num3;
    }
```

面试题35：第一个只出现一次的字符

题目：在字符串中找出第一个只出现一次的字符。如输入“abaccdeff”,则输出‘b’

```
public static char firstNotRepeatingChar(String s){
        if(s == null || "".equals(s))return '\0';
        int[] tmp = new int[256];
        Arrays.fill(tmp, 0);
        for(int i = 0; i < s.length(); ++i){
                if(tmp[s.charAt(i)] != 0){
                        tmp[s.charAt(i)] += 1;
                }else{
                        tmp[s.charAt(i)] = 1;
                }
        }
        for(int i = 0; i < tmp.length; i++){
                if(tmp[i] == 1){
                        return (char)i;
                };
        }
        return '\0';
}
```