题目：

Merge *k* sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

就是合并两个排序链表的加强版

1.遍历一遍所有的链表然后同时把元素都取出来然后保存再Arraylist中，然后对Arraylist进行排序，用Collections.sort（list）进行排序。也可以用数组保存，然后用Arrays.sort（array）进行排序。同时扩展一下，倒序呢？自己再编写一个新的排序类继承Comparator接口，实现接口中的compare方法。详情见学习文档。

public static class Com implements Comparator{

@Override

public int compare(Object o1, Object o2) {

if((int)o1>(int)o2)

return -1;

if((int)o1<(int)o2)

return 1;

else

return 0;

}

}

public static ListNode mergeKLists(ListNode[] lists) {

ListNode head = new ListNode();

ListNode p = head;

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

for(ListNode q : lists){

q=q.next;

while(q!=null){

nums.add(q.val);

q=q.next;

}

}

Collections.reverse(nums.new Com()); //倒序，正序去掉后边的参数即可

for (int i : nums){

ListNode q = new ListNode(i);

p.next=q;

p=q;

}

return head;

}

2.递归。每次合并一个数列然后进行排序，递归n次正好全部合并。正好用到了之前编写的合并两个链表的函数。参数一定要加一个退出递归的条件，这个方法的推出条件就是ListNode[]的length，从头开始合并就是递归开始的时候n为0，每次调用的函数n的参数值+1，到n=length-1的时候推出返回合并的ListNode。从尾开始就是n=length，每次n-1，在n=0的时候退出返回结果。

public static ListNode mergeKListsByRecursion(ListNode result,ListNode[] lists,int n){

ListNode reshead = new ListNode();

ListNode resnode = reshead;

if(n==0) { //递归的结束

return result;

}

else {

ListNode p = lists[n-1].next;

ListNode q = result.next;

while (p != null && q != null) {

if (p.val < q.val) {

resnode.next = p;

p = p.next;

} else {

resnode.next = q;

q = q.next;

}

resnode=resnode.next;

}

if (q == null)

resnode.next = p;

else

resnode.next = q;

}

return mergeKListsByRecursion(reshead,lists,n-1); /\*递归，每次参数n-1，直到n=0

的时候返回res。在写递归的时候一般有返回值的，在出口return要返回的值在非出口就返回自己递归的函数就行了。如果没有返回值就直接调用递归函数\*/

}

递归开始在main{

ListNode = mergerKListByRecursion(new ListNode(),lists,lists.length);

}