Question 1

(a) (i)

随问题规模n的增大，算法执行时间的增长率和f(n)的增长率相同，

称作算法的渐进时间复杂度，简称时间复杂度。其中f(n)是问题规模n的某个函数。

时间复杂度不包括这个函数的低阶项和首项系数

(ii)

7n2+9n+5

：

项的阶数有：7n^2>9n>5

去除低阶项得7n^2

去除首项系数为：n^2

即结果为：O(n^2)

(b)

初始状态，将待排序数组分为已排序和未排序两部分，然后，遍历，将未排序序列中的元素逐一插入到已排序的序列中。

复杂度：

平均复杂度：第一步遍历整个数组n，然后对于每一个数，插入到之前的已排序数组，需要对比已排序数组n，即复杂度为n^2

最好复杂度：当元是数组已经有序，则只需要遍历一次，每次只对比一个数，复杂度为n=n\*1

(c)

不可计算函数即：对于一个问题，无法给出确切的解。

说明：有比如：有一个函数集合B（即这个集合B是函数f的集合），并且B不是空集，也不是全部函数f，那么随机程序P是否属于B是不可计算的。

Question 2

(a)

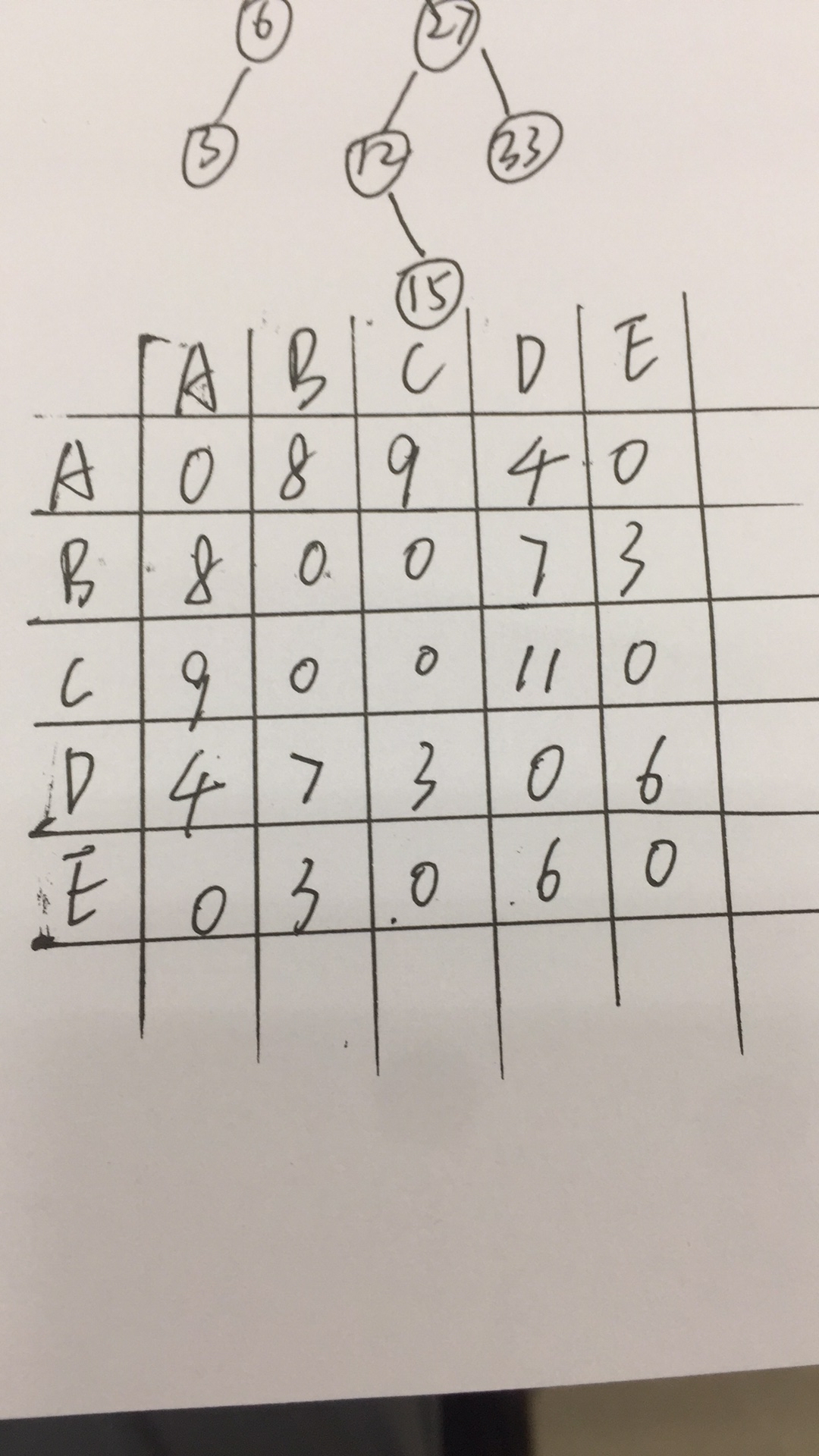
public int getNumOfNonLetters(BinaryTree<String> bt) {  
 int num = 0; // 总字符串数  
 if (bt.isEmpty()) {  
 // 如果为空，则字符数为0，返回  
 return num;  
 } else {  
 // 递归调用，加上左右节点总数  
 num = getNumOfNonLetters(bt.leftChild())+1;  
 num = num+getNumOfNonLetters(bt.rightChild());  
 }  
 return num;  
}

(b)

public class CharStackImpl implements CharStack {  
 private int size; // 大小  
 private Node head; // 头结点  
  
 public CharStackImpl() { //无参构造函数  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return size == 0;  
 }  
  
 @Override  
 public void push(char c) {  
 Node current = head;  
 head = new Node();  
 head.value = c;  
 head.next = current;  
 size++;  
 }  
  
 @Override  
 public char top() {  
 // 返回头结点  
 if (head == null) {  
 throw new StackException();  
 }  
 return head.value;  
 }  
  
 @Override  
 public void pop() {  
 // 移除头结点  
 if (head == null) {  
 throw new StackException();  
 }  
 head = head.next;  
 size--;  
 }  
  
 class Node {// 存储数据的自定义linkedlist  
 private char value;  
 private Node next;  
 }  
  
 class StackException extends RuntimeException {  
 public StackException() {  
 super("Stack is null");  
 }  
 }  
}

Question 3

(a)邻接矩阵



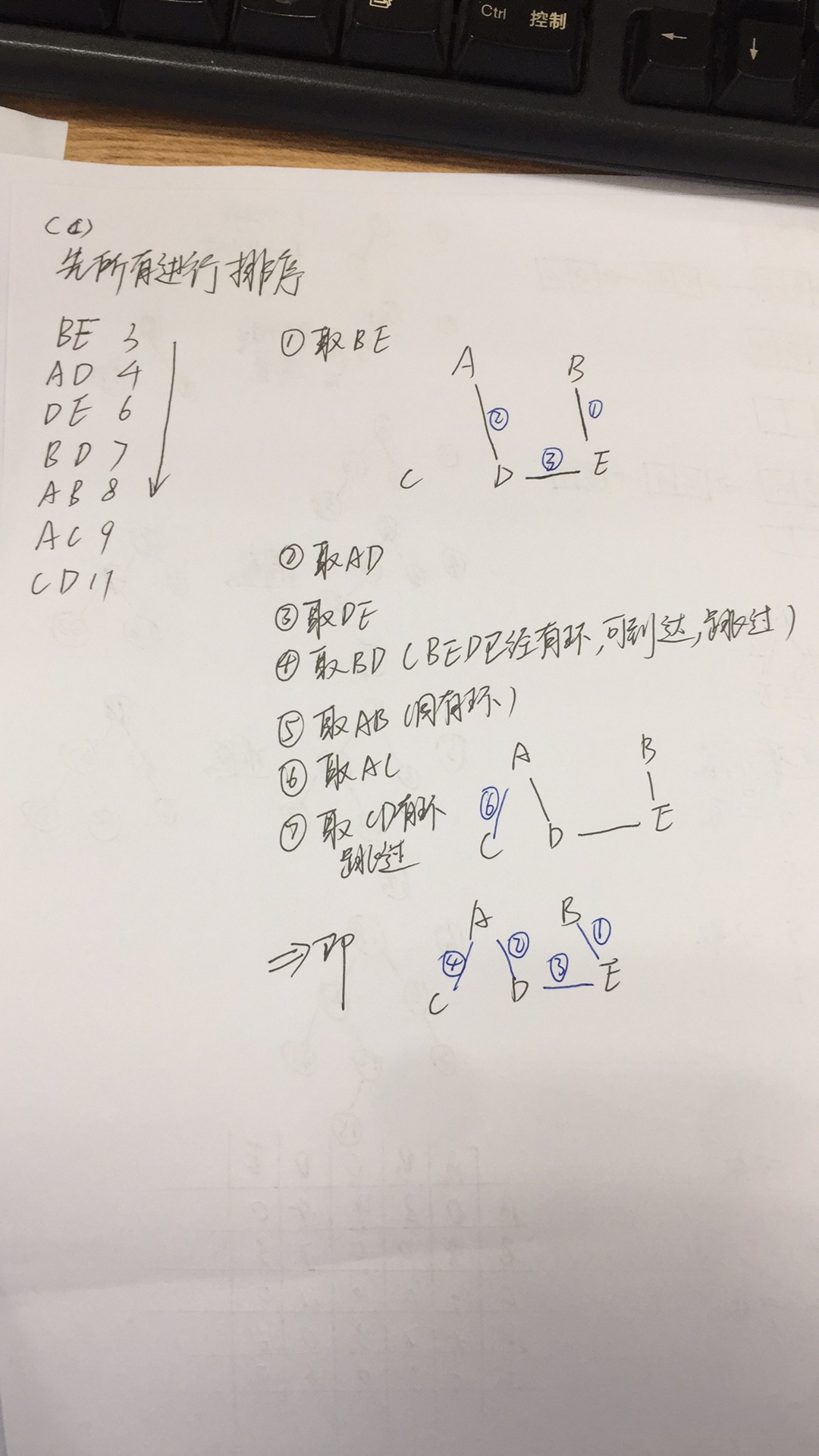
(b)

1.将图各边按照权值进行排序

2.将图遍历一次，找出权值最小的边，（条件：此次找出的边不能和已加入最小生成树集合的边构成环），若符合条件，则加入最小生成树的集合中。不符合条件则继续遍历图，寻找下一个最小权值的边。

3.递归重复步骤1，直到找出n-1条边为止（设图有n个结点，则最小生成树的边数应为n-1条），算法结束。得到的就是此图的最小生成树。

(c)

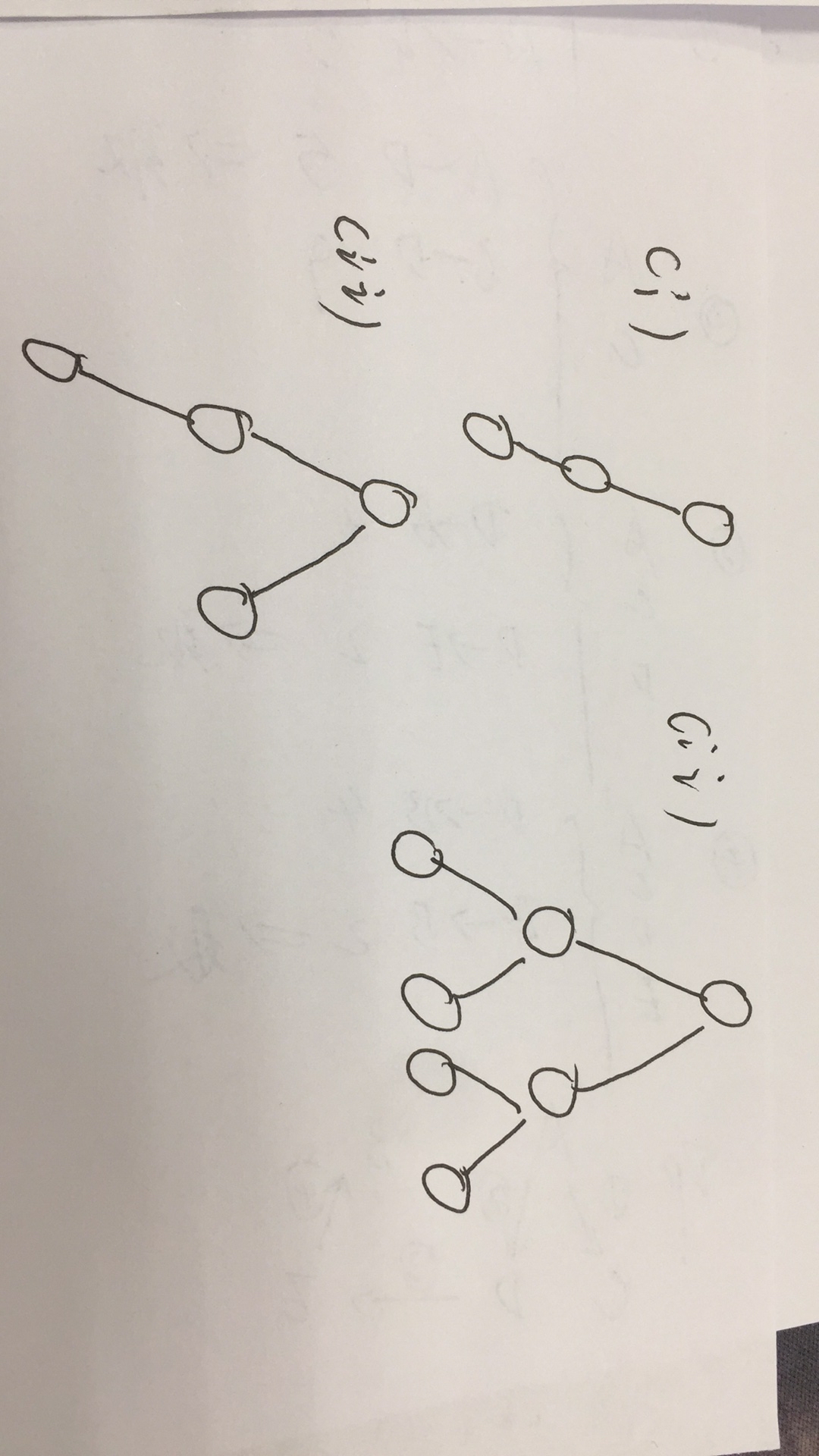


Question 4

(a)

平衡二叉树要求对于每一个节点来说，它的左右子树的高度之差不能超过1，如果插入或者删除一个节点使得高度之差大于1，就要进行节点之间的旋转，将二叉树重新维持在一个平衡状态。

(b)



(c)

