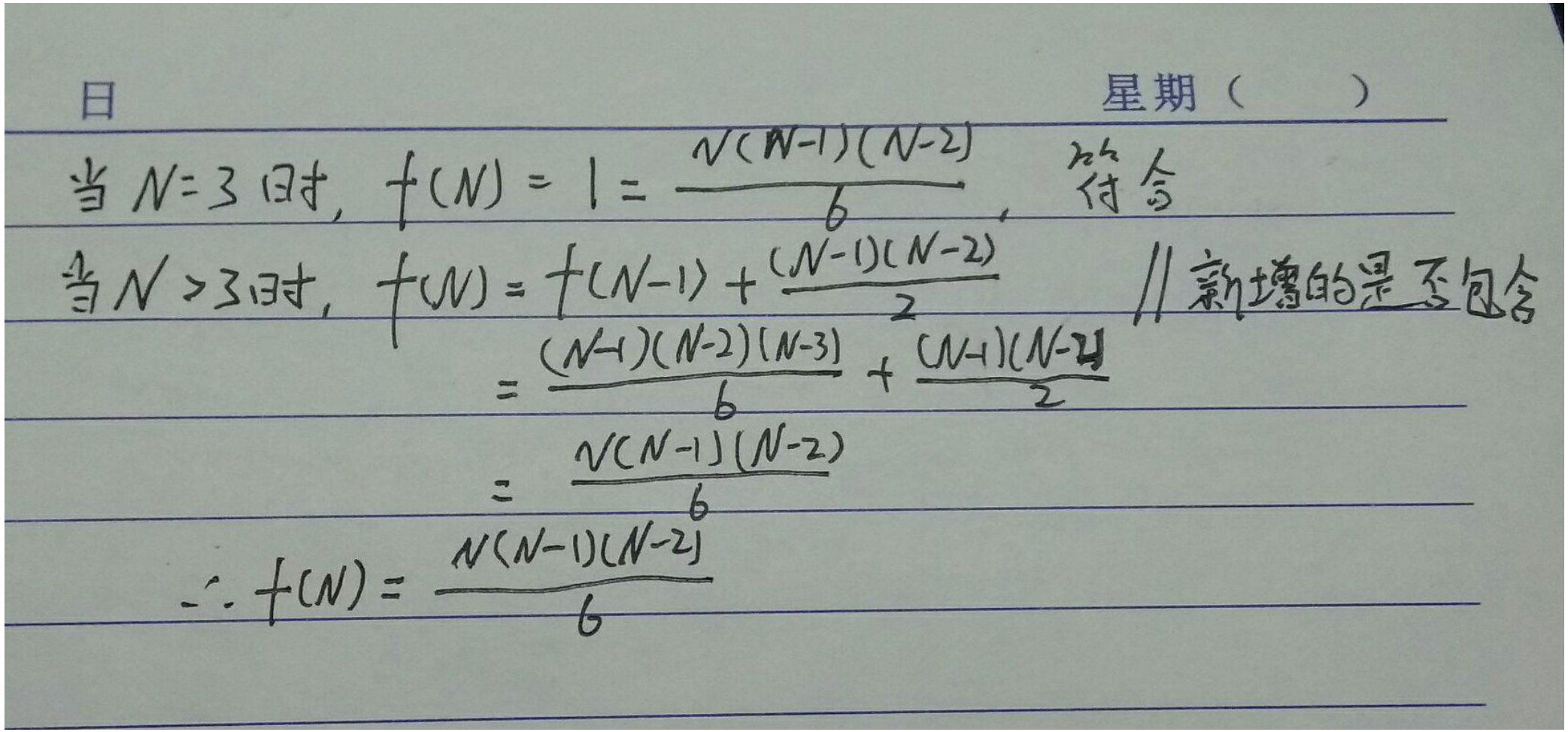
1.4.1 证明从N个数中取三个整数的不同组合的总数为N(N-1)(N-2)/6。提示：使用数学归纳法。



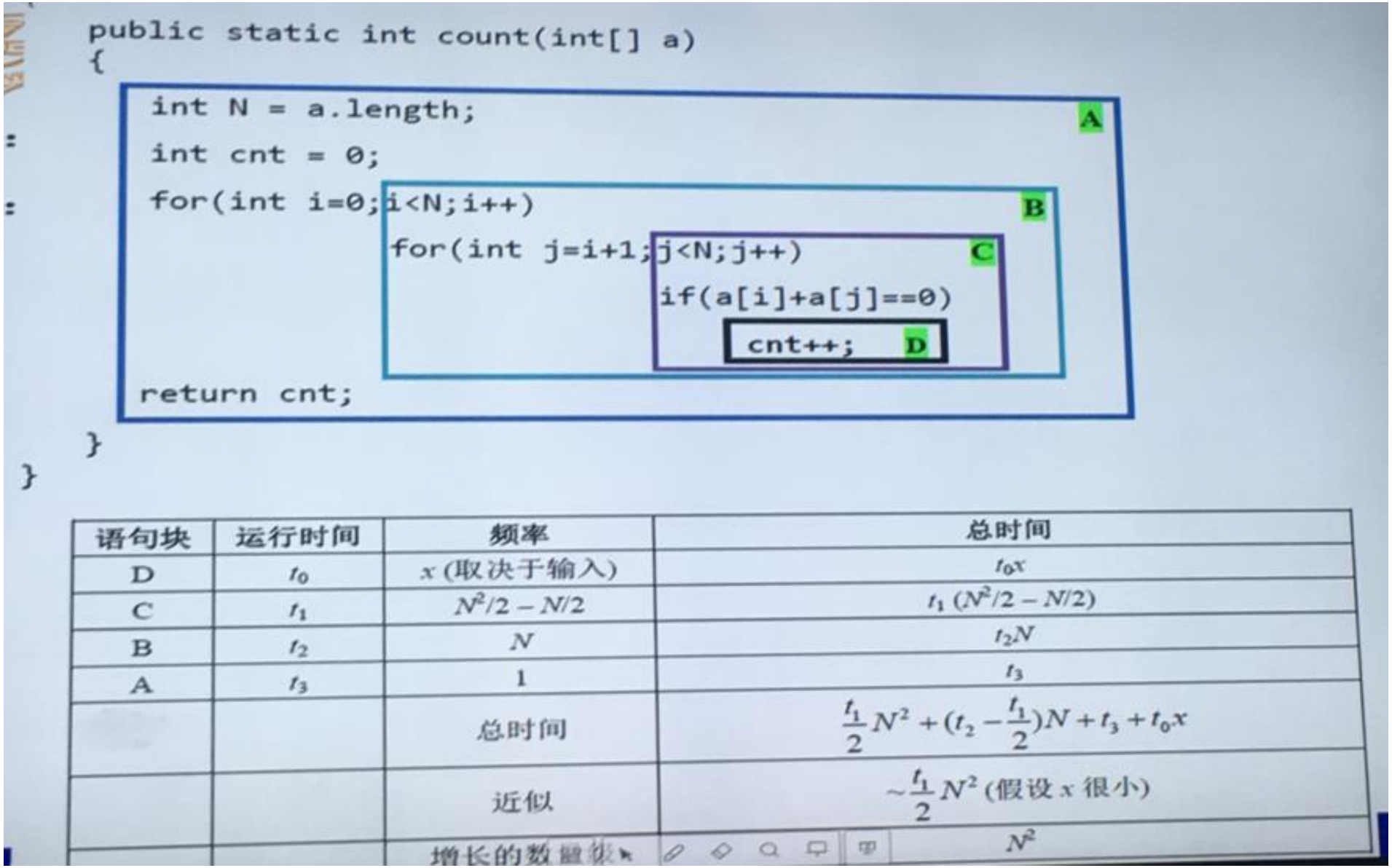
1.4.2 修改ThreeSum，正确处理两个较大的int值相加可能溢出的情况。

```
int N = a.length;
int cnt = 0;
for(int i=0;i<N;i++)
    for(int j=i+1;j<N;j++)
        for(int k=j+1;k<N;k++)
        {
            min=min(a[i],a[j],a[k]);
            mid=mid(a[i],a[j],a[k]);
            max=max(a[i],a[j],a[k]);
            if(mid>0 && min+mid==-max)
                cnt++;
            else if(mid<0 && max+mid==-min)
                cnt++;
            else if(mid==0 && max=-min)
                cnt++;
        }
```

或者

```
If (min < 0 && max > 0)
{
   If (min + max == -mid)
        cnt++;
}
```

1.4.4 参照表1.4.4为TwoSum建一张类似的表格。



```
public static int count(int[] a)
{
   int N = a.length;                                    A
   int cnt = 0;
   for(int i=0;i<N;i++)                                 B
                for(int j=i+1;j<N;j++)                  C
                     if(a[i]+a[j]==0)
                          cnt++;                        D
   return cnt;
}
}
```

语句块	运行时间	频率	总时间
D	t_0	x (取决于输入)	$t_0 x$
C	t_1	$N^2/2 - N/2$	$t_1 (N^2/2 - N/2)$
B	t_2	N	$t_2 N$
A	t_3	1	t_3
		总时间	$\frac{t_1}{2}N^2 + (t_2 - \frac{t_1}{2})N + t_3 + t_0 x$
		近似	$\sim\frac{t_1}{2}N^2$ (假设 x 很小)
		增长的数量级	N^2

1.4.5

题目： 给出下面这些量的近似。

解答：

a. $N + 1 \sim N$

b. $1 + 1/N \sim 1$

c. $(1 + 1/N)(1 + 2/N) \sim 1$

d. $2N^3 - 15N^2 + N \sim 2N^3$

e. $\lg(2N)/\lg(N) \sim 1$

f. $\lg(N^2+1)/\lg(N) \sim 2$

g. $N^{100}/2^N \sim 1/2^N$

提示： 最后一小题中，2^N是指数级，当N很大时， 2^N > N^100

1.4.6 给出以下代码的运行时间的增长数量级（作为N的函数）。

```
for (int n = N; n > 0; n /= 2)
    for(int i = 0; i < n; i++)
        sum++;
```

b.
```
int sum = 0;
for (int i = 1; i < N; i *= 2)
    for (int j = 0; j < i; j++)
        sum++;
```

c.
```
int sum = 0;
for (int i = 1; i < N; i *= 2)
    for (int j = 0; j < N; j++)
        sum++;
```

a. $N(1 + 1/2 + 1/4 + 1/8 + \ldots + 1/N)$
$= N(2 - 1/N) = 2N - 1 \sim 2N$

b. $(1 + 2 + 4 + 8 + \ldots + N) = 2N - 1 \sim 2N$

c. $N \lg N$

1.4.7

题目： 以统计涉及输入数字的算术（和比较）操作的成本模型分析ThreeSum。

解答： 如图 1.4.4 所示，设加法的成本为 t_0，比较的成本为 t_1。

则在 E 语句块中的成本为 t_0x，x 取决于输入。

在 D 语句块中，每次循环共有两次比较，三次加法，

所以总的成本为$(2t_1 + 3t_0)(N^3/6 - N^2/2 + N/3)$。

在 C 语句块中共有 2 次加法，1 次比较，时间成本为$(t_1 + 2t_0)(N^2/2 - N/2)$。

在 B 语句块中共有 2 次加法，1 次比较$(t_1 + 2t_0)N$。

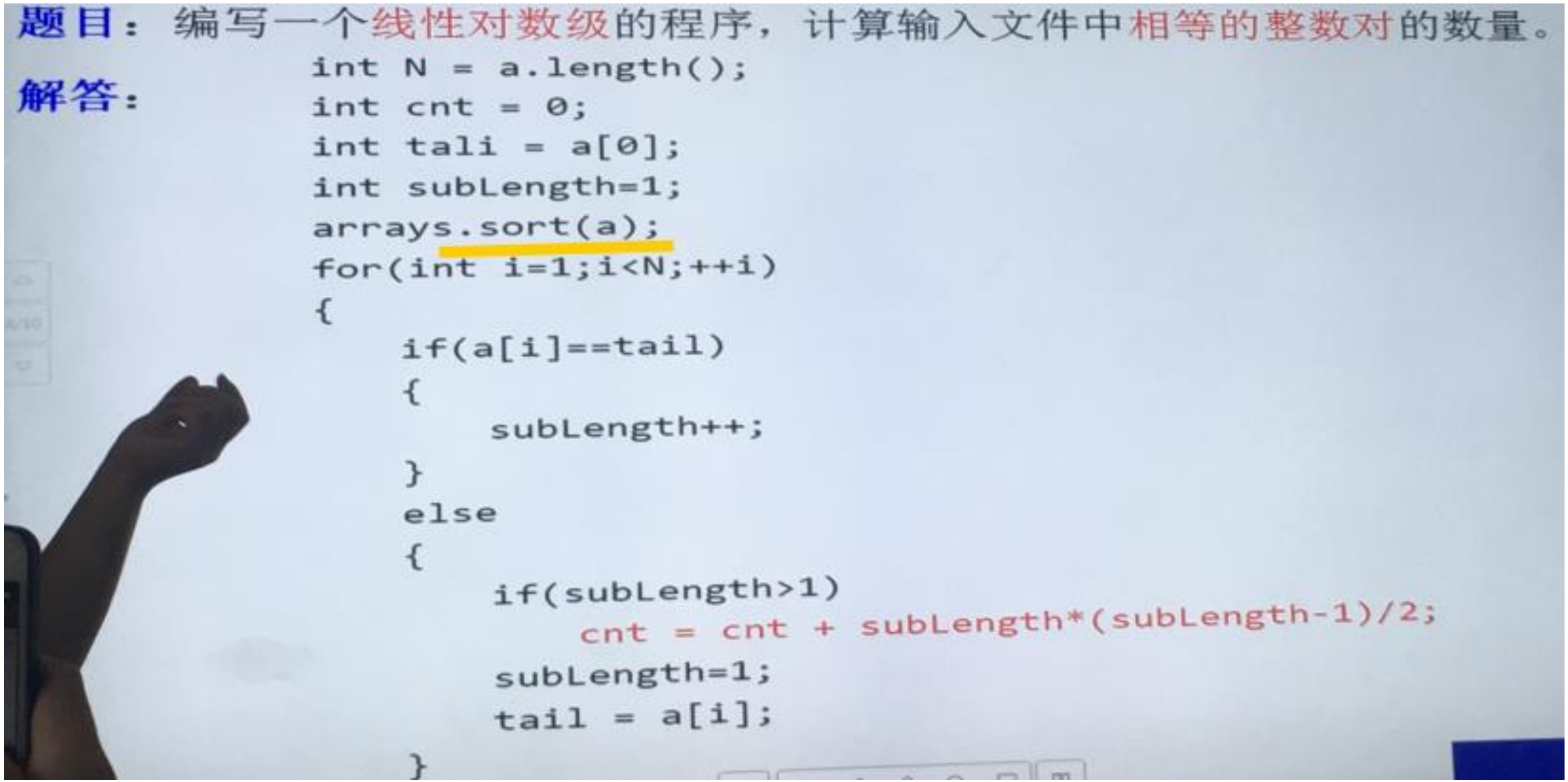
类似的在 A 语句块总共成本为 t_2(求 a.length 的成本)。

所以，总的成本为四者累加为:

$$(2t_1 + 3t_0)(N^3/6 - N^2/2 + N/3) + (t_1 + 2t_0)(N^2/2 - N/2) + (t_1 + 2t_0)N + t_0x + t_2$$

令 $t_0 = 1$，$t_1 = 1$，渐近时间复杂度为$5N^3/6$

1.4.8


题目：编写一个线性对数级的程序，计算输入文件中相等的整数对的数量。
解答：
int N = a.length();
int cnt = 0;
int tali = a[0];
int subLength=1;
arrays.sort(a);
for(int i=1;i<N;++i)
{
if(a[i]==tail)
{
subLength++;
}
else
{
if(subLength>1)
cnt = cnt + subLength*(subLength-1)/2;
subLength=1;
tail = a[i];
}

1.4.15

题目： 快速3-sum。用一个线性级别的算法实现TwoSumFaster来计算有序的数组中和为0的整数对的数量。用相同的思想为3-sum问题给出平方级别的算法。

解答： 用两个指针分别指向数组的起始位置和末尾位置，求二者之和，小于0，前指针++，大于0后指针--，等于0，输出，二指针同时移动。指针指向同一位置时结束。

```
arrays.sort(a);
int head=0;
int tail=a.length()-1;
cnt=0;
while(head<tail)
{
    int sum=a[head]+a[tail];
    if(sum>0)
        --tail;
    else if(sum<0)
        ++head;
    else
    {
        ++head;
        --tail;
        ++cnt;
    }
}
return cnt;
```

1.5.1

题目： 使用quick-find算法处理序列9-0 3-4 5-8 7-2 2-1 5-7 0-3 4-2。对应输入的每一对整数，给出id[]数组的内容和访问数组的次数。

解答：

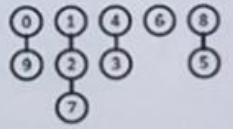
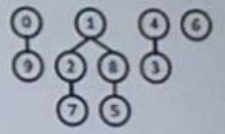
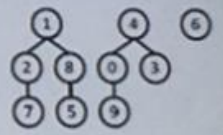
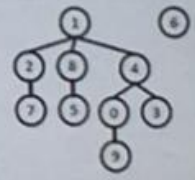
数对(p-q)	id 数组内容	数组访问次数
9-0	0 1 2 3 4 5 6 7 8 0	15
3-4	0 1 2 4 4 5 6 7 8 0	15
5-8	0 1 2 4 4 8 6 7 8 0	15
7-2	0 1 2 4 4 8 6 2 8 0	15
2-1	0 1 1 4 4 8 6 1 8 0	16
5-7	0 1 1 4 4 1 6 1 1 0	16
0-3	4 1 1 4 4 1 6 1 1 4	16
4-2	1 1 1 1 1 1 6 1 1 1	18

数组访问次数的计算方法如下。处理每一个数对(p-q)时，调用connected(p, q)需要2次读操作，如果p和q当前没有连接(本题中每一对输入均是未连接的)，需要调用union(p, q)：union(p, q)执行时，获取p和q的id需要2次读操作，for循环需要10次读操作并有x次写操作(x对应于数组中改变的位置个数)。因此，处理一个数对(p-q)的数组访问次数为：2+2+10+x。

1.5.2

题目： 使用quick-union算法处理序列9-0 3-4 5-8 7-2 2-1 5-7 0-3 4-2。对应输入的每一对整数，给出id[]数组的内容和访问数组的次数，并画出森林。

解答：

数对(p-q)	id 数组内容	数组访问次数	森林
9-0	0123456780	5	
3-4	0124456780	5	
5-8	0124486780	5	
7-2	0124486280	5	
2-1	0114486280	5	
5-7	0114486210	11	
0-3	4114486210	7	
4-2	4114186210	7	

处理每一个数对(p-q)时，首先需要调用connected(p, q)，执行时需要调用一次root(p)和一次root(q)，如果p和q当前没有连接(本题中每一对输入均是未连接的)，需要调用union(p, q)；

union(p, q)执行时，也需要调用一次root(p)和一次root(q)，同时执行一次写操作。

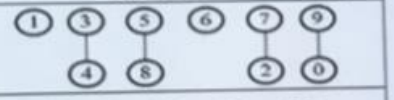
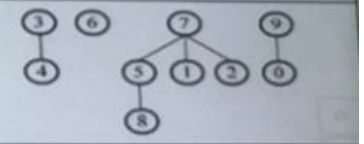
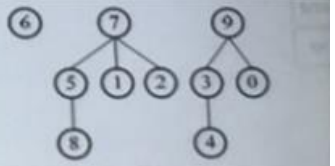
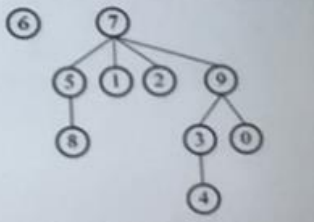
执行一次root(node)需要访问数组的次数为node的树深。

因此，处理一个数对(p-q)的数组访问次数为：$2(depth_p + depth_q) + 1$，其中$depth_{node}$表示node的树深。

1.5.3

题目： 使用加权quick-union算法处理序列9-0 3-4 5-8 7-2 2-1 5-7 0-3 4-2。对应输入的每一对整数，给出id[]数组的内容和访问数组的次数，并画出森林。

解答：

数对(p-q)	id 数组内容	数组访问次数	森林
9-0	9123456789	8	
3-4	9123356789	8	
5-8	9123356759	8	
7-2	9173356759	8	
2-1	9773356759	10	
5-7	9773376759	8	
0-3	9779376759	10	
4-2	9779376757	14	

处理每一个数对(p-q)时，首先需要调用connected(p, q)，执行时需要调用一次root(p)和一次root(q)，如果p和q当前没有连接(本题中每一对输入均是未连接的)，需要调用union(p, q)；

union(p, q)执行时，也需要调用一次root(p)和一次root(q)，同时执行2次sz[]数组的读操作、1次id[]数组的写操作和1次sz[]数组的写操作。

执行一次root(node)需要访问数组的次数为node的树深。

因此，处理一个数对(p-q)的数组访问次数为：$2(depth_p + depth_q) + 4$，其中$depth_{node}$表示node的树深。

注意：这里给出的是计算所有数组访问次数的方法。

1.5.8

题目： 用一个反例证明quick-find算法中的union()方法的以下直观实现是错误的：

```
public void union(int p, int q)
{
        if (connected(p, q)) return;
        for (int i = 0; i < id.length; i++)
                if (id[i] == id[p]) id[i] = id[q];
        count--;
}
```

Answer. The value of id[p] changes to id[q] in the for loop. Thus, any object r > p with id[r] equal to id[p] will not be updated to equal id[q].

答：在 for 循环中，id[p]将会被赋值为 id[q]。这样，对于任意的满足 r > p 且 id[r]=id[p]的 id[r]将不会被更新为 id[q]。

1.5.10

题目： 在加权quick-union算法中，假设我们将id[find(p)]的值设为q而非id[find(q)]，所得的算法是正确的吗？

解答： 所得算法是正确的。

但是，这样做是把p所在树的根连接到了q，而非连接到q所在树的根。这会增加树的高度，因此无法保证同样的性能。

2.2.2

题目： 按照算法2.4所示的轨迹的格式给出自顶向下的归并排序是如何将数组E A S Y Q U E S T I O N排序的。

解答：

```
mid = lo + (hi - lo)/2;
sort(a, lo, mid);
sort(a, mid+1, hi)
```

lo	m	hi	a[ ] 0	1	2	3	4	5	6	7	8	9	10	11
			E	A	S	Y	Q	U	E	S	T	I	O	N
0	0	1	A	E	S	Y	Q	U	E	S	T	I	O	N
0	1	2	A	E	S	Y	Q	U	E	S	T	I	O	N
3	3	4	A	E	S	Q	Y	U	E	S	T	I	O	N
3	4	5	A	E	S	Q	U	Y	E	S	T	I	O	N
0	2	5	A	E	Q	S	U	Y	E	S	T	I	O	N
6	6	7	A	E	Q	S	U	Y	E	S	T	I	O	N
6	7	8	A	E	Q	S	U	Y	E	S	T	I	O	N
9	9	10	A	E	Q	S	U	Y	E	S	T	I	O	N
9	10	11	A	E	Q	S	U	Y	E	S	T	I	N	O
6	8	11	A	E	Q	S	U	Y	E	I	N	O	S	T
0	5	11	A	E	E	I	N	O	Q	S	S	T	U	Y
			A	E	E	I	N	O	Q	S	S	T	U	Y

2.2.3

题目： 按照算法2.4所示的轨迹的格式给出自底向上的归并排序是如何将数组E A S Y Q U E S T I O N排序的。

解答：

lo	m	hi	a[] 0	1	2	3	4	5	6	7	8	9	10	11
			E	A	S	Y	Q	U	E	S	T	I	O	N
0	0	1	A	E	S	Y	Q	U	E	S	T	I	O	N
2	2	3	A	E	S	Y	Q	U	E	S	T	I	O	N
4	4	5	A	E	S	Y	Q	U	E	S	T	I	O	N
6	6	7	A	E	S	Y	Q	U	E	S	T	I	O	N
8	8	9	A	E	S	Y	Q	U	E	S	I	T	O	N
10	10	11	A	E	S	Y	Q	U	E	S	I	T	N	O
0	1	3	A	E	S	Y	Q	U	E	S	I	T	N	O
4	5	7	A	E	S	Y	E	Q	S	U	I	T	N	O
8	9	11	A	E	S	Y	E	Q	S	U	I	N	O	T
0	3	7	A	E	E	Q	S	S	U	Y	I	N	O	T
0	7	11	A	E	E	I	N	O	Q	S	S	T	U	Y
			A	E	E	I	N	O	Q	S	S	T	U	Y

课堂补充： 只有当n为2的整数次幂时自顶向下和自底向上归并排序所对应的子问题才完全一样，只是合并函数执行的过程不一样。

2.2.4

题目：是否当且仅当两个输入的子数组都有序时归并排序才能得到正确的结果？证明你的结论，或者给出一个反例。

解答：是的。原地归并排序一定需要两个待排序的数组都是有序的，否则结果就是错的。

比如：数组A：2, 1, 3, 4

数组B：1, 2, 3, 4

结果是：1, 2, 1, 2, 3, 3, 4, 4，明显是错的。

2.2.5

题目： 当输入数组的大小$N=39$时，给出自顶向下和自底向上的归并排序中各次归并子数组的大小及顺序。

解答：

```
mid = lo + (hi - lo)/2;
sort(a, lo, m);
sort(a, mid+1, hi)
```

省略比 2 或 3 小的子问题

自顶向下的归并排序:

3,2,5, 3,2,5, 10, 3,2,5, 3,2,5, 10, 20
3,2,5, 3,2,5, 10, 3,2,5, 2,2,4, 9, 19
39

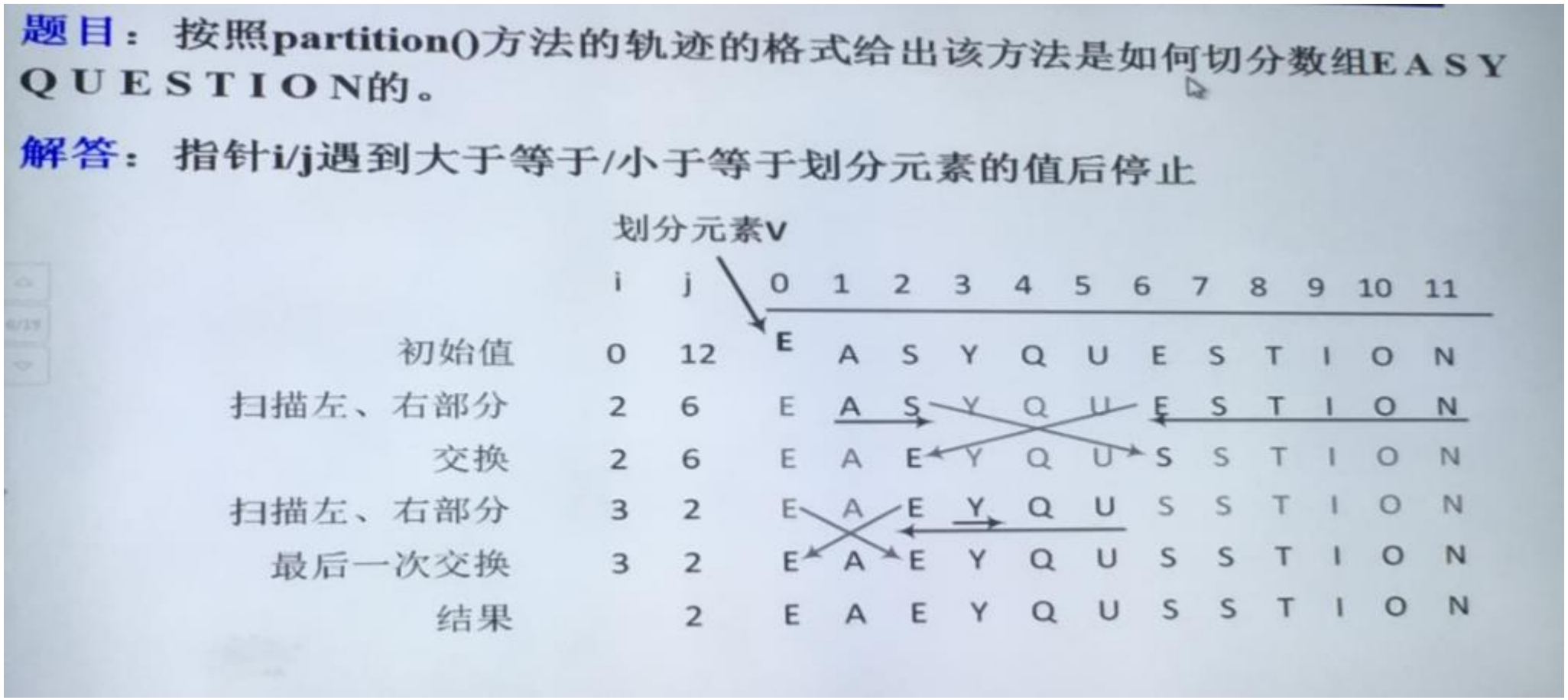
自底向上的归并排序:

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1,
4, 4, 4, 4, 4, 4, 4, 4, 4, 3,
8, 8, 8, 8, 7,
16, 16, 7
32, 7

2.3.1

题目： 按照partition()方法的轨迹的格式给出该方法是如何切分数组E A S Y Q U E S T I O N的。

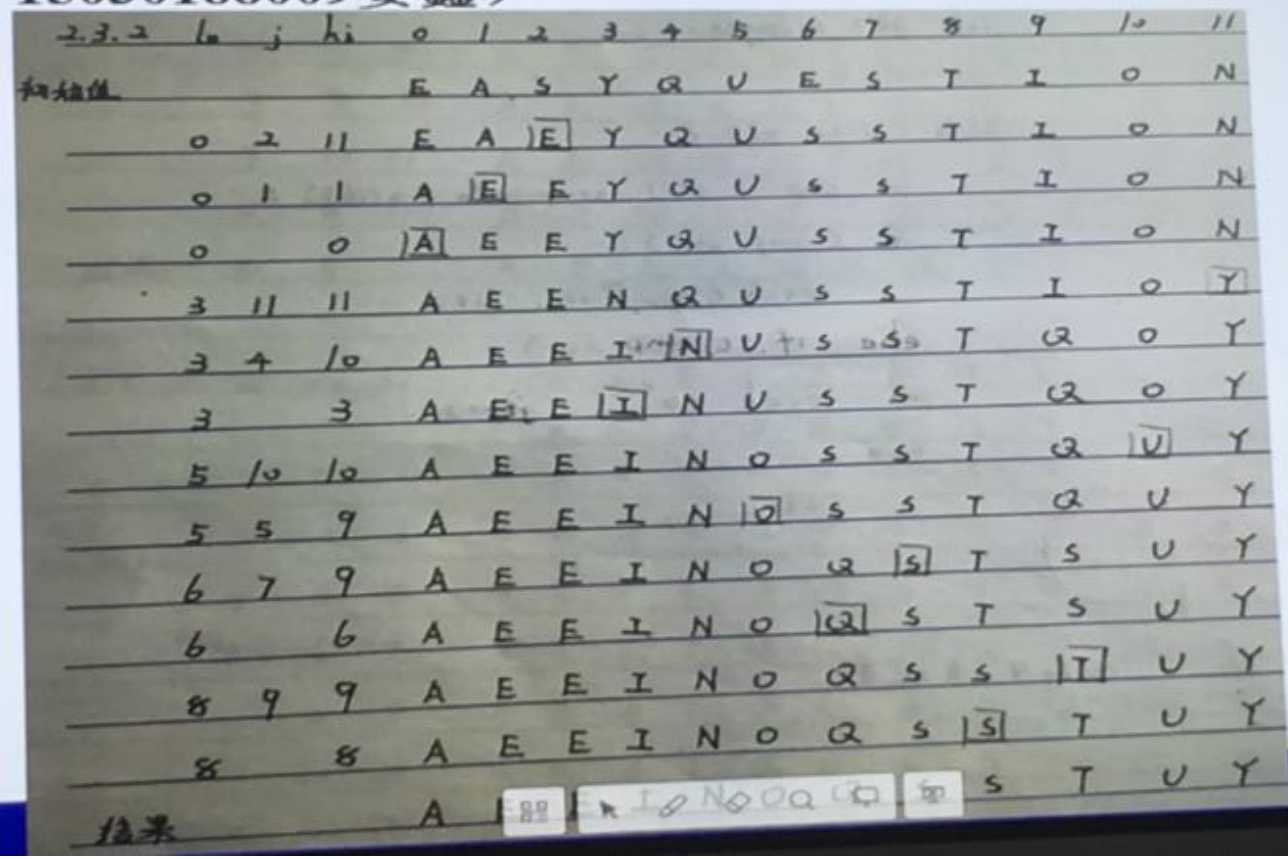
解答： 指针i/j遇到大于等于/小于等于划分元素的值后停止



2.3.2

题目： 按照本节中快速排序所示轨迹的格式给出快速排序是如何将数组EASY QUESTION排序的（出于练习的目的，可以忽略开头打乱数组的部分）。

解答： （from 15030188009姜鑫）



2017/10/23

2.3.3

题目： 对于长度为N的数组，在Quick.sort()执行时，其最大的元素最多会被交换多少次？

解答： $N/2$。 （from 16030147003 段龙）

假设输入的 n 个元素为 1~n。

现构造这 n 个元素形成的排列，使得快速排序时最大元素被交换 $n/2$ 次：

2 n 4 1 6 3 8 5 (9+1) (10-3) ... (i+1) (i+1-3) ... 最后两/一个元素

此排列分成三部分：

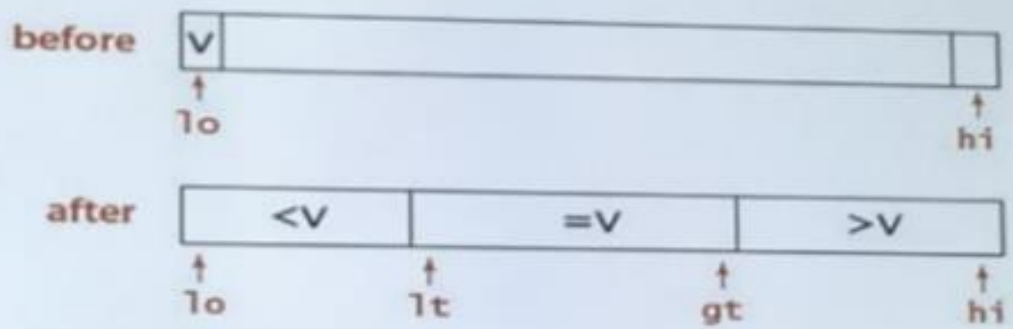
第一部分，即前两个元素，固定为 2 n；

第二部分，第 i 和第 i+1 位置上的元素分别为 i+1 和 i+1-3；

2.3.5

题目：给出一段代码将已知两种主键值的数组排序。

解答：对输入的数组执行一次三向划分操作，这样就完成了只含两个键值的数组的排序。



注意：本题中，执行一次普通的划分操作，不能完成排序。因为在将划分元素排定之后，只能保证划分元素左边的元素小于等于划分元素，右边的元素大于等于划分元素。

比如，输入ABAABABBBAB，经历一次划分将得到AAAABABBBBB。

2.3.8

题目：Quick.sort()在处理N个全部重复的元素时大约需要多少次比较？

解答：~$N\lg N$次比较。

备注：当数组中元素都相同时，指针i和j每前进一步就交换一次指向的元素，每次划分将数组均分成两部分。

2.3.9

题目： 请说明Quick.sort()在处理只有两种主键值的数组时的行为，以及在处理只有三种主键值的数组时的行为。

解答： 标准的快速排序在处理只有两种或三种主键值的数组时，整体的时间性能都是线性对数级别的。

具体地，在处理只有两种主键值的数组时：第一次划分后，将a[j]排定，同时会将a[j]左边或右边的所有元素都排定；对于排定的部分后续的划分操作是不必要的，对于没有排定的部分继续重复第一次划分的情况。

处理只有三种主键值的数组时，分析方法类似。

利用三向划分的快速排序可以高效地处理含有大量重复元素的数组，可以将排序时间从线性对数级降低到线性级。

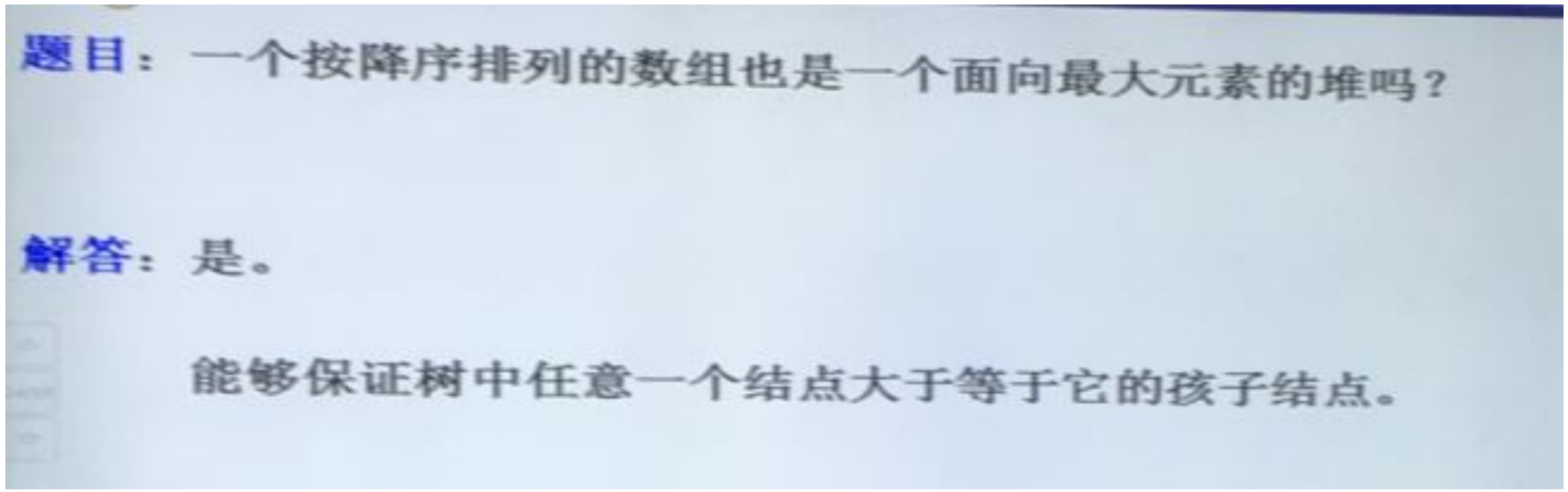
2.4.3

题目： 用以下数据结构实现优先队列，支持插入元素和删除最大元素的操作：无序数组、有序数组、无序链表和链表。将你的4种实现中每种操作在最坏情况下的运行时间（上限）制成一张表格。

解答：

方法	无序数组	有序数组	无序链表	有序链表
插入元素	1	N	1	N
删除最大元素	N	1	N	1

2.4.4


题目：一个按降序排列的数组也是一个面向最大元素的堆吗？
解答：是。
能够保证树中任意一个结点大于等于它的孩子结点。

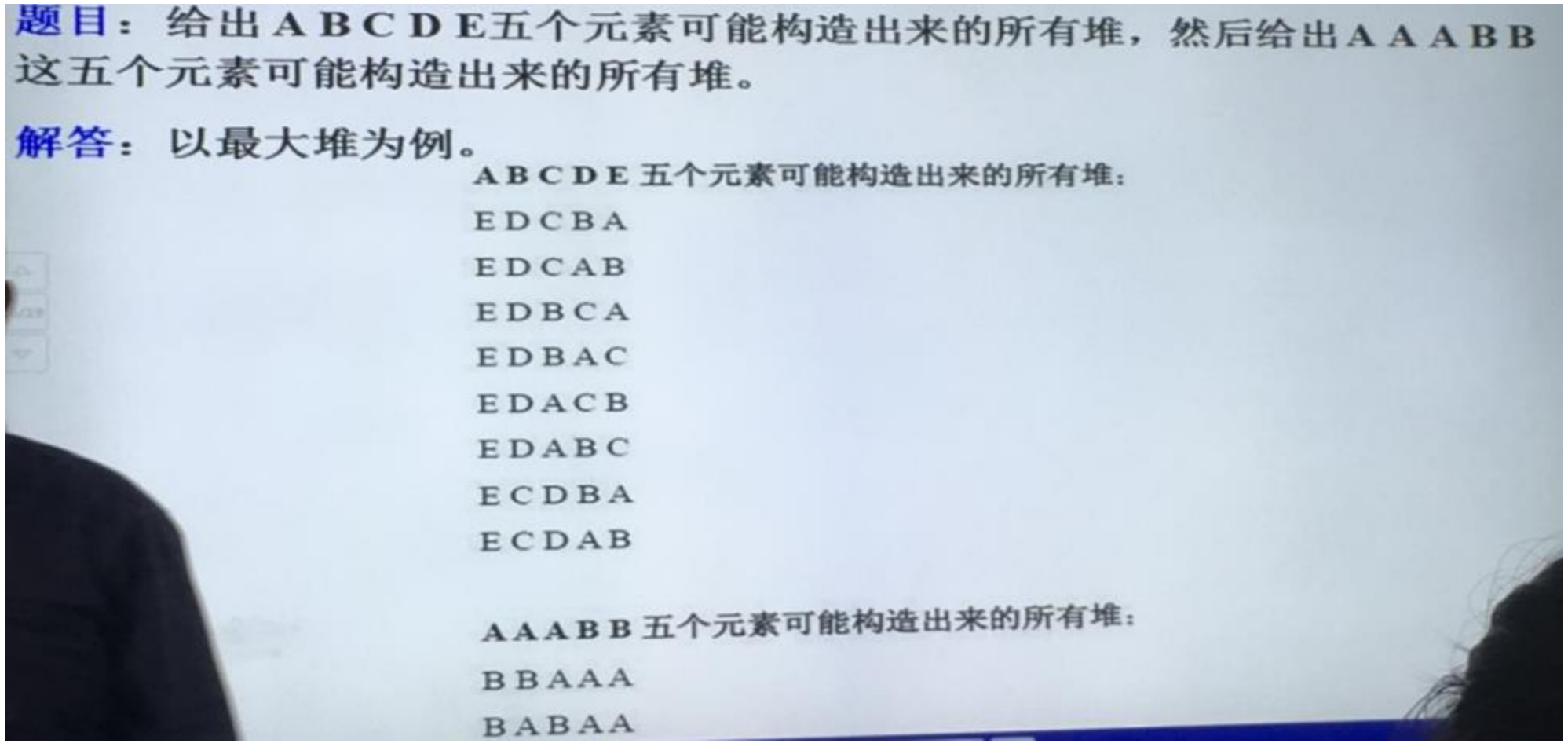
2.4.6

题目： 按照练习2.4.1的规则，用序列PRIO*R**I*T*Y***QUE***U*E操作一个初始为空的面向最大元素的堆，给出每次操作后堆的内容。

解答：

操作	堆的内容
插入P	P
插入R	RP
插入I	RPI
插入O	RPIO
删除最大元素	POI
插入R	RPIO
删除最大元素	POI
删除最大元素	OI
插入I	OII
删除最大元素	II
插入T	TII
删除最大元素	II
插入Y	YII
删除最大元素	II
删除最大元素	I
删除最大元素	
插入Q	Q
插入U	UQ
插入E	UQE
删除最大元素	QE
删除最大元素	E
删除最大元素	
插入U	U
删除最大元素	
插入E	E

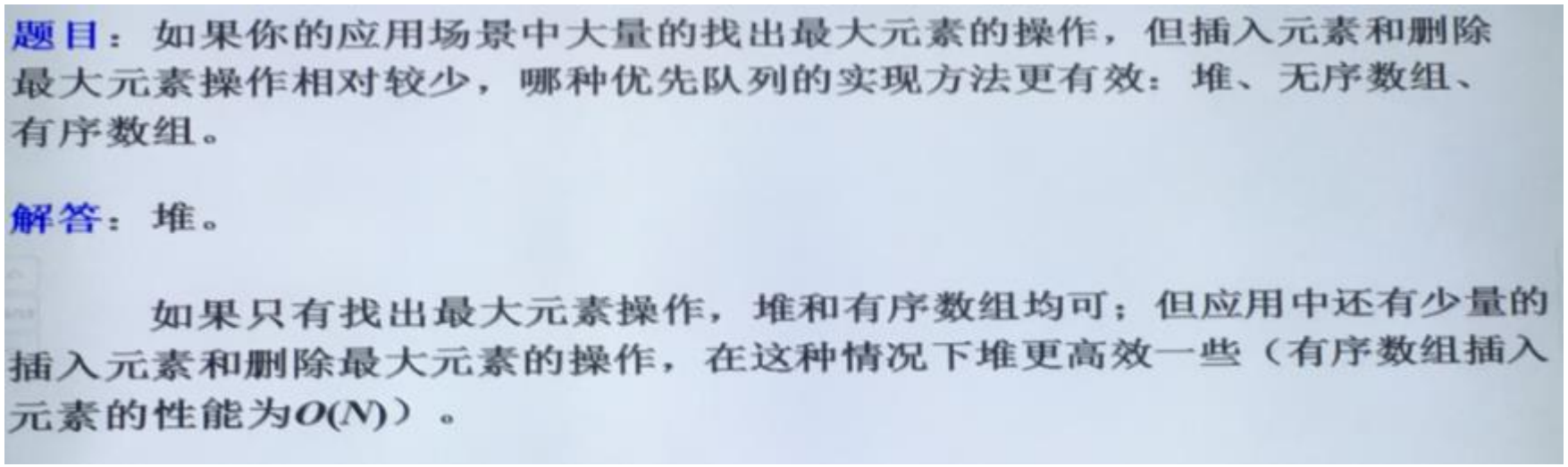
2.4.9


题目：给出ABCDE五个元素可能构造出来的所有堆，然后给出AAABB这五个元素可能构造出来的所有堆。
解答：以最大堆为例。
ABCDE五个元素可能构造出来的所有堆：
EDCBA
EDCAB
EDBCA
EDBAC
EDACB
EDABC
ECDBA
ECDAB
AAABB五个元素可能构造出来的所有堆：
BBAAA
BABAA

2.4.11 如果你的应用中有大量的插入元素的操作，但只有若干删除最大元素操作，哪种优先队列的实现方法更有效：堆、无序数组、有序数组？

无序数组。

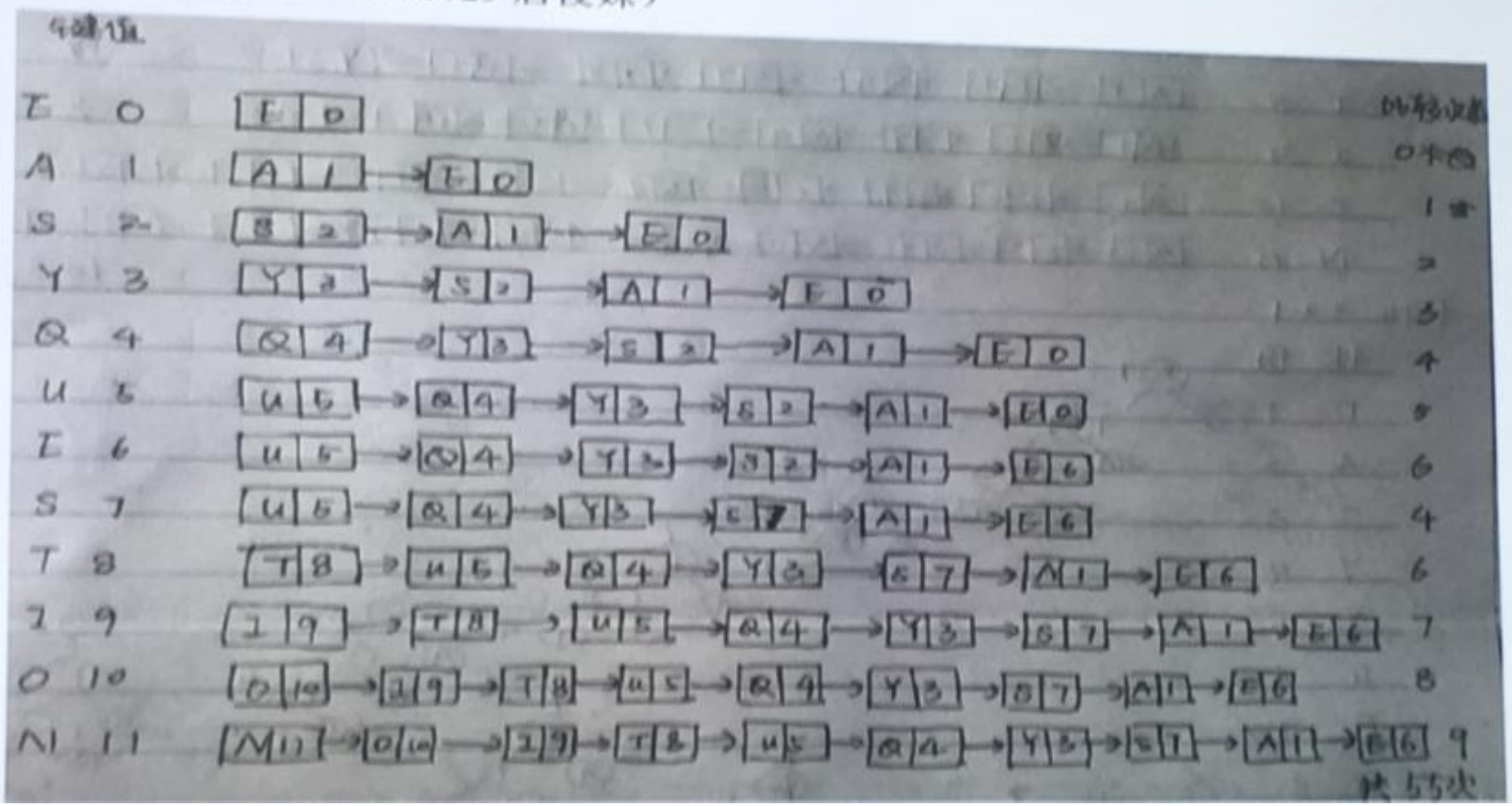
2.4.12



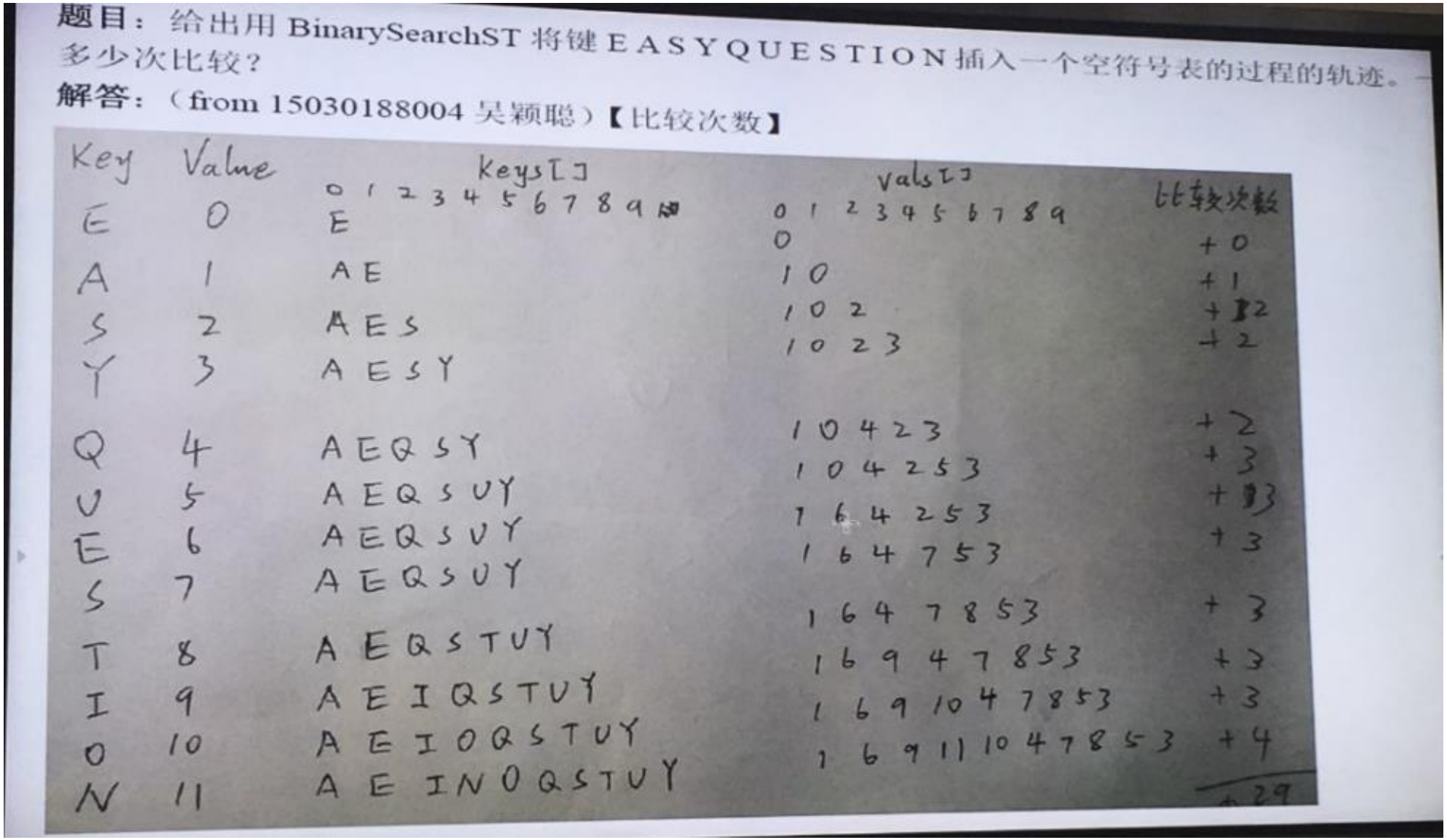
3.1.10

题目：给出用 SequentialSearchST 将键 E A S Y Q U E S T I O N 插入一个空符号表的过程的轨迹。一共进了多少次比较？

解答：（from 15030188029 唐俊姝）



3.1.11



题目： 给出用 BinarySearchST 将键 E A S Y Q U E S T I O N 插入一个空符号表的过程的轨迹。一多少次比较？

解答：（from 15030188004 吴颖聪）【比较次数】

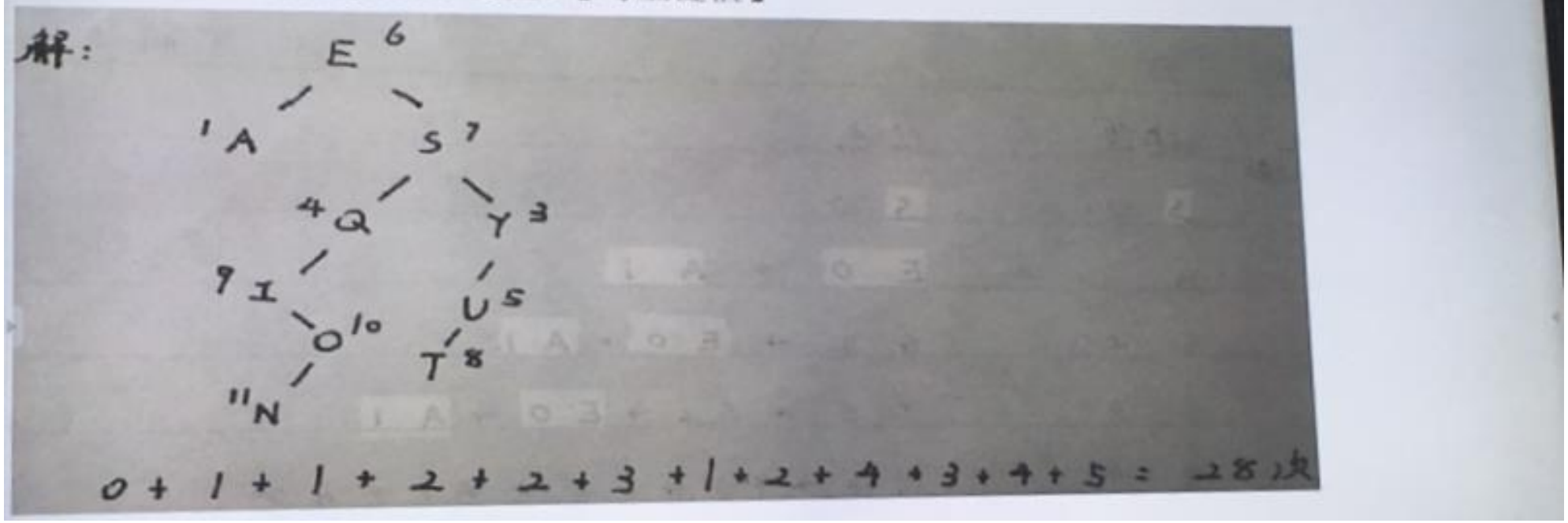
Key	Value	keys[] 0 1 2 3 4 5 6 7 8 9	vals[] 0 1 2 3 4 5 6 7 8 9	比较次数
E	0	E	0	+0
A	1	A E	1 0	+1
S	2	A E S	1 0 2	+2
Y	3	A E S Y	1 0 2 3	+2
Q	4	A E Q S Y	1 0 4 2 3	+2
U	5	A E Q S U Y	1 0 4 2 5 3	+3
E	6	A E Q S U Y	1 6 4 2 5 3	+3
S	7	A E Q S U Y	1 6 4 7 5 3	+3
T	8	A E Q S T U Y	1 6 4 7 8 5 3	+3
I	9	A E I Q S T U Y	1 6 9 4 7 8 5 3	+3
O	10	A E I O Q S T U Y	1 6 9 10 4 7 8 5 3	+3
N	11	A E I N O Q S T U Y	1 6 9 11 10 4 7 8 5 3	+4
				+29

3.2.1

题目： 将EASYQUESTION作为键按顺序插入一棵初始为空的二叉查找树中（方便起见设对应的值为i），画出生成的二叉查找树。构造这棵树需要多少次比较？

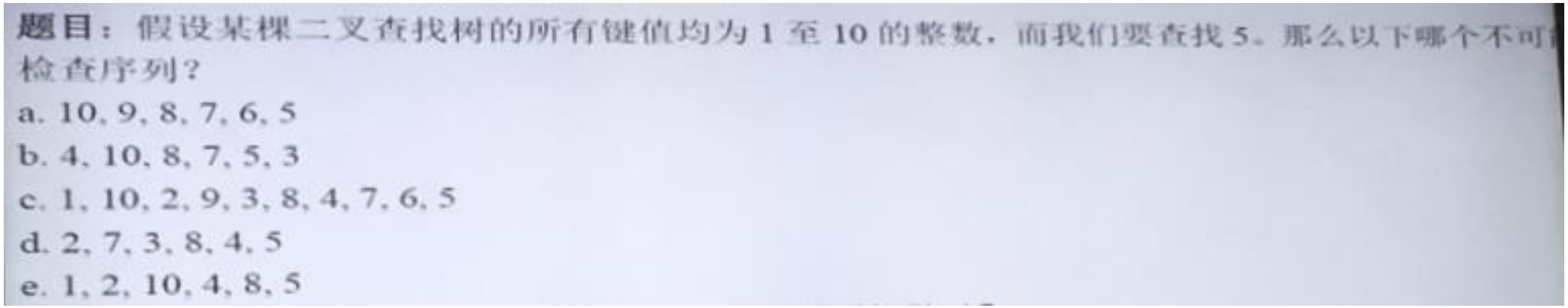
解答：（from 15030188009 姜鑫）【写上键值】

解：



0 + 1 + 1 + 2 + 2 + 3 + 1 + 2 + 4 + 3 + 4 + 5 = 28次

3.2.4

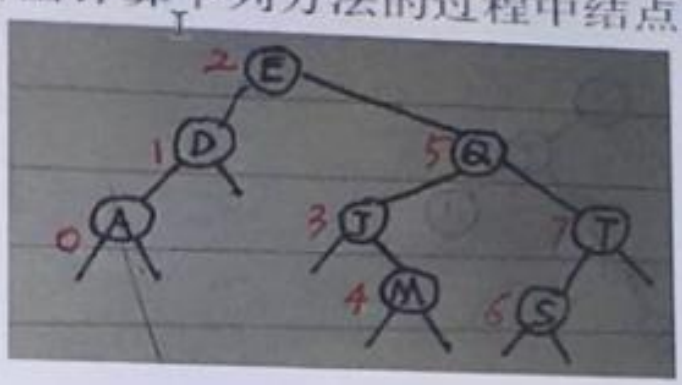


a可能
b不可能
c可能
d不可能
e可能

3.2.15

题目：对于右下方的二叉查找树，给出计算下列方法的过程中结点的访问序列。

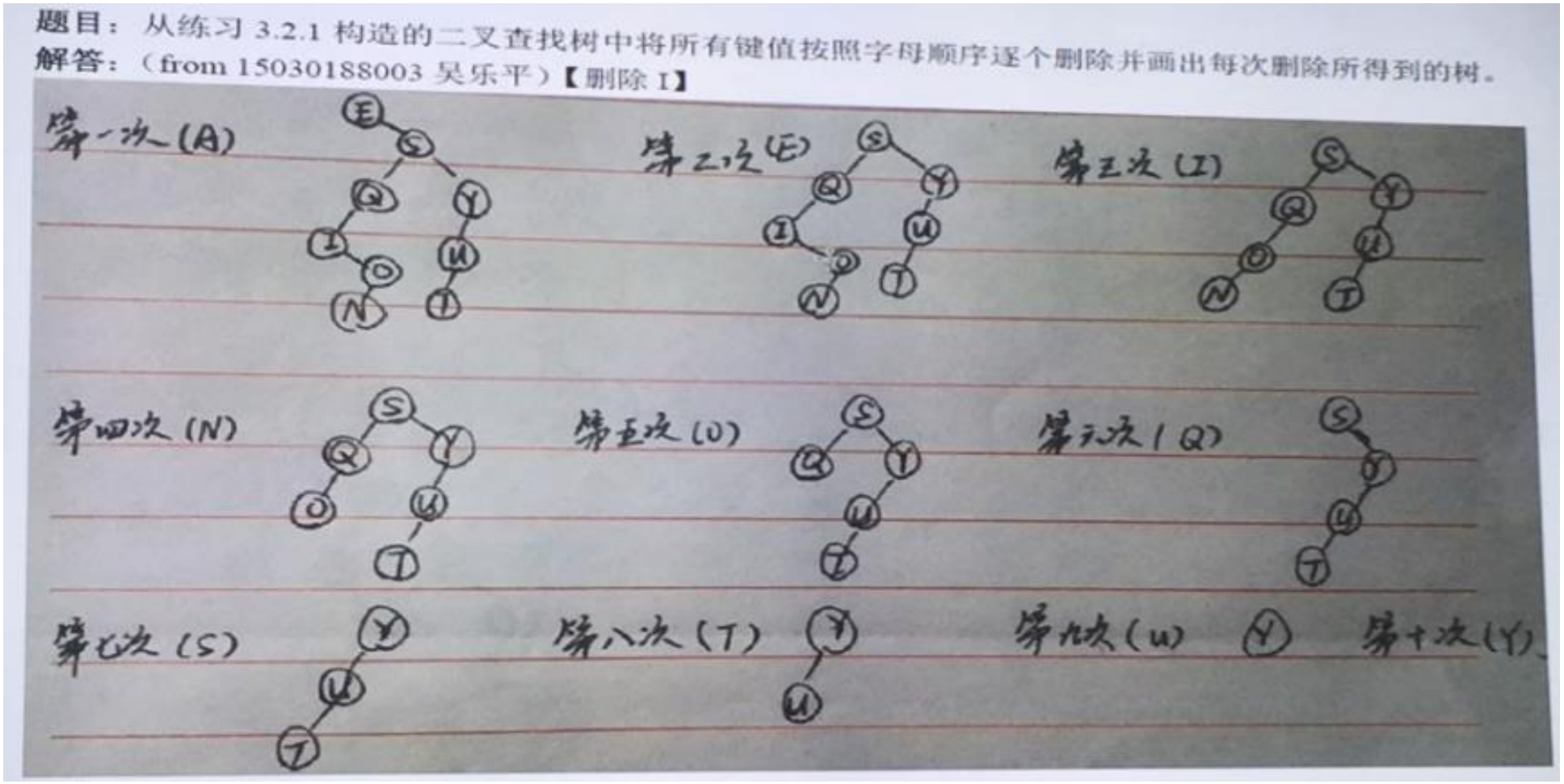
a. floor("Q") E Q
b. select(5) E Q
c. ceiling("Q") E Q
d. rank("J") E Q J
e. size("D", "T") 略
f. keys("D", "T") 略



3.2.18

题目：从练习 3.2.1 构造的二叉查找树中将所有键值按照字母顺序逐个删除并画出每次删除所得到的树。

解答：（from 15030188003 吴乐平）【删除 I】



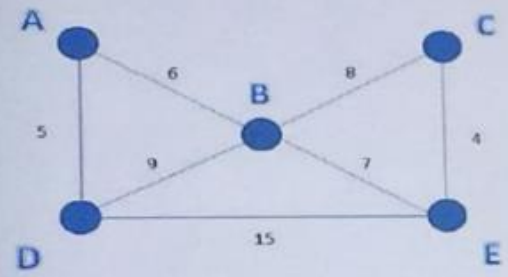
4.3.3 证明当图中所有边的权重均不同时，图的最小生成树唯一。

假设它存在两颗最小生成树a, b
我们找到这两个方案不同的边中最小的一条边x, (既x在一个生成树上，不在另一个生成树上)。
假设x在a里面，那我们考虑把x塞到b里面去，这样b+x里面肯定有一个环，
环上至少有一条边比x大（如果每条边都比x小，那根据x的定义这些边肯定也都在a里面，那a就有环了，矛盾），
把这条边去掉之后还是一个生成树（你考虑把环去掉一条边变成了链，但是原来联通的现在也还是联通的）。
但是这颗生成树比b小（加进了一条x，去掉了一条比x大的边），所以b不是最小生成树，矛盾
故最小生成树是唯一的

4.3.13

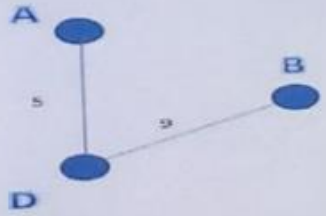
题目：给出一个反例证明以下策略不一定能够找到图的最小生成树：首先以任意顶点作为图的最小生成树，然后向树中添加 V − 1 条边，每次总是添加依附于最近加入最小生成树的顶点的所有边中的权重最小者。

答： 例：

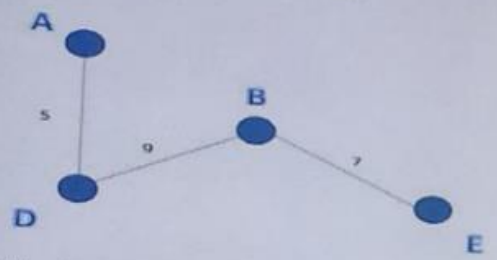


(1) A 为任意顶点，先找到 A-D；

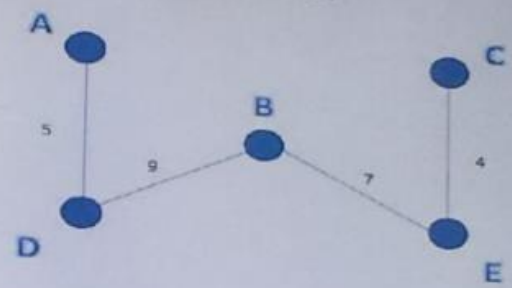
(2) 再以 D 为顶点找到 D-B；



(3) 再以 B 为顶点找到 B-E；

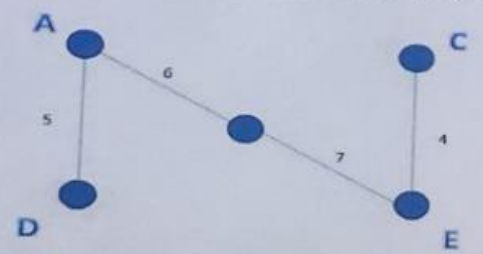


(4) 再以 E 为顶点找到 E-C；



结果：这样总权重为 5+9+7+4=25；

但是存在以下情况，总权重为 5+6+7+4=22<25；



4.4.1

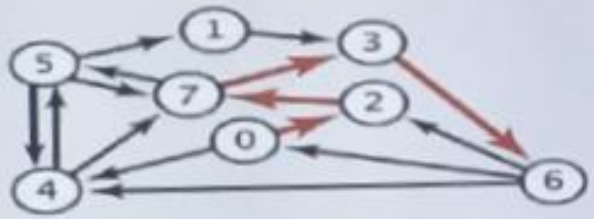
题目： 真假判断：将每条边的权重都加上一个常数不会改变单点最短路径问题的答案。
答： 假。与路径的长度有关。

4.4.5

题目： 在 tinyEWD.txt 中（请见图 4.4.4）改变边 0 -> 2 的方向。画出该加权有向图中以顶点 2 为同的最短路径树。【两棵？】

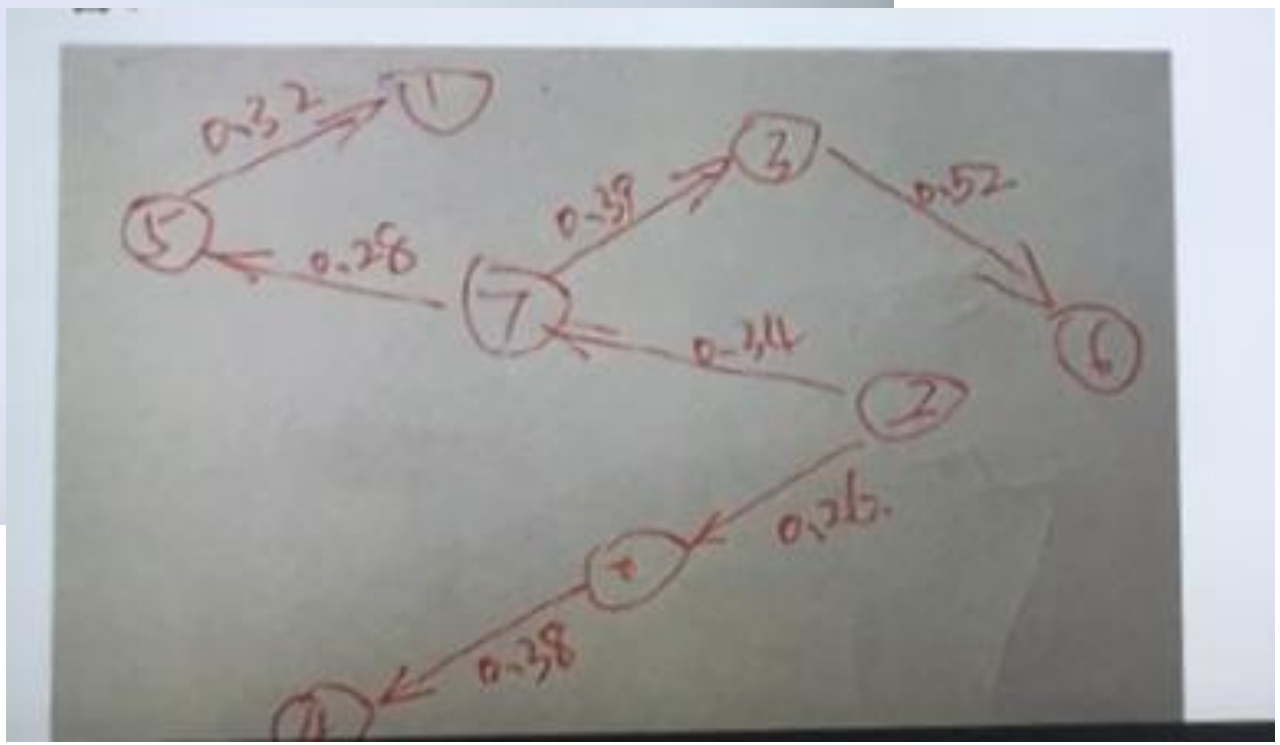
edge-weighted digraph

4->5	0.35
5->4	0.35
4->7	0.37
5->7	0.28
7->5	0.28
5->1	0.32
0->4	0.38
0->2	0.26
7->3	0.39
1->3	0.29
2->7	0.34
6->2	0.40
3->6	0.52
6->0	0.58
6->4	0.93



shortest path from 0 to 6

0->2	0.26
2->7	0.34
7->3	0.39
3->6	0.52



4.4.9

题目： 表 4.4.10 来自于一张很早以前出版的公路地图，它显示的是城市之间的最短路径的长度。这张表中有一个错误。改正这个错误并建立一张表来说明最短路径是哪条。

	Providence	Westerly	New London	Norwich
Providence	-	53	54	48
Westerly	53	-	18	101
New London	54	18	-	12
Norwich	48	101	12	-

答：

Norwich 与 Westerly 之间的最短路径有错误。应该更新为：

Norwich - New London - Westerly，路程为：12 + 18 = 30。