1.判断

(a) F

(b) T

(c) F

(d) F

(e) F

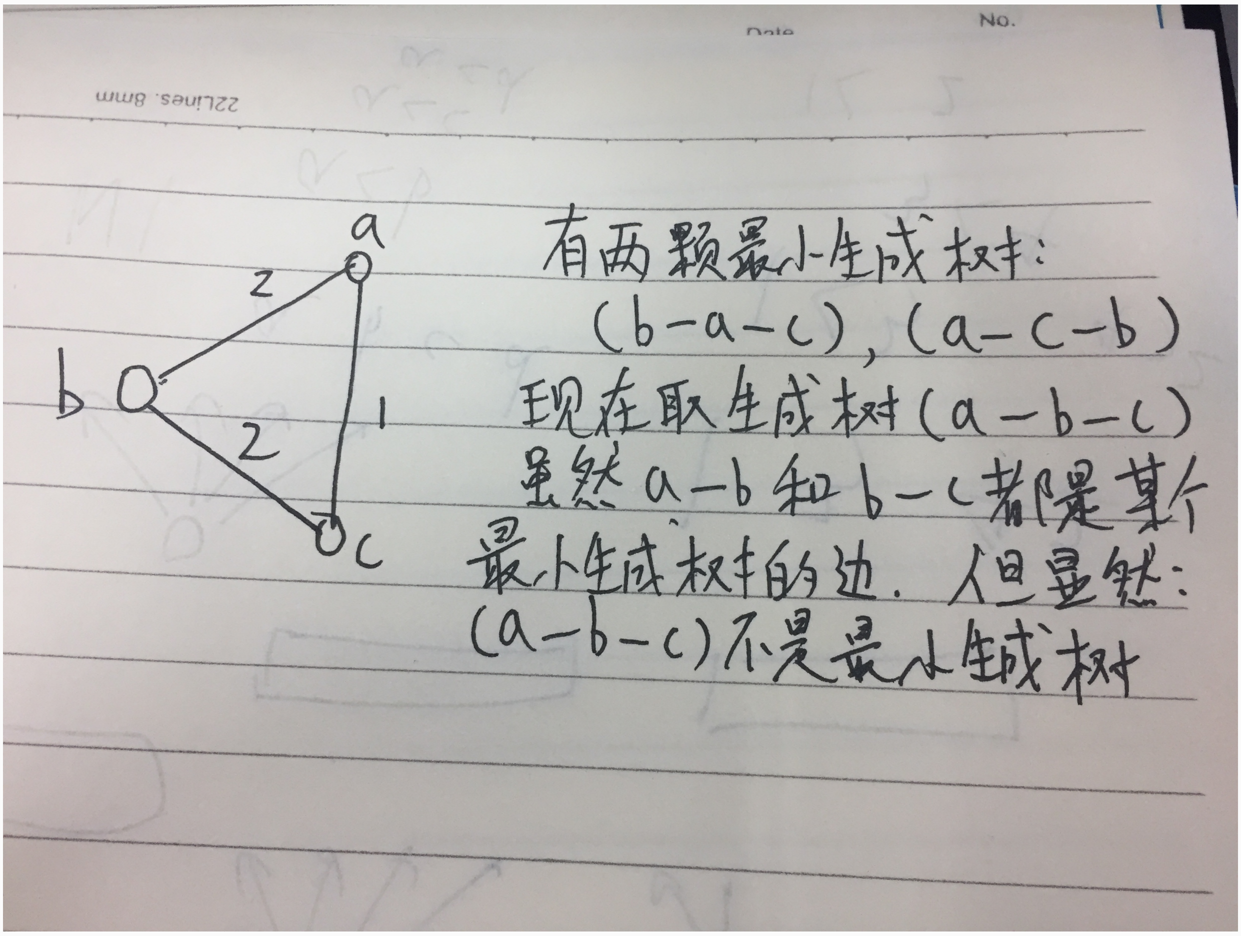
2.简答

(a) $2^{\sqrt{2lgn}} < \sqrt{n} < nlgn < n^2 < (lgn)!$

(b) http://blog.csdn.net/little_newbee/article/details/52188152?locationnum=9&fps=1

(c) a = 7, b = 3. 由于$n^{(log_3 7)} < n^2$，根据主定理，$T(n) = \Theta(n^2)$

(d) 不能。 反例：



3.贪心

> - 首先，根据Si从左到右进行排序

- 从i=1开始，我们放置第一个基站。第一个基站的左端点刚好与S1重合.也就是第一个基站覆盖范围为[S1,S1+2r].
- 假设第一个基站覆盖了1~j户人家，那么我们放置第二个基站：左端点与Sj+1重合。也就是第二个基站覆盖范围为[Sj+1,Sj+1 + 2r].
- 以此类推，知道所有人家全被基站覆盖。此时所用基站最少。

证明：因为要覆盖所有人家，所以任何情况下最左侧的基站覆盖范围的左端点L一定要<=S1。现在我们考虑L<S1的情况，若L<S1，那么该区间覆盖的人家数一定小于等于L=S1的情况。 如果L<S1的情况下覆盖人家数确实变少了，那么这组解肯定是不优的了，因为剩余的人家数变多了。 如果L<S1的情况下覆盖人家数和L=S1的情况下覆盖人家数一样，那么L<S1的情况下最终解也最多是和L=S1一样优，不会更优。对于剩下没被覆盖到的点，我们也用同样的方式考虑。因此证明完毕。

4.分治

```
// O(n)时间将有序的A和B合并成一个有序的L，并计算A和B之间的逆序数、夸张逆序数
Merge-and-Count(A,B)
初始化 指针p1指向A的第一个元素,p2指向B的第一个元素
初始化 L为空表
初始化 n = m = 0
while p1和p2都还指向一个元素:
    IF A[p1] < B[p2]:
        将A[p1] 加到L末尾
        p1++
    ELSE:
        将B[p2] 加到L末尾
        p2++
        n += A中p1到表尾的元素个数
将A或B剩余元素加到L末尾

重新初始化 指针p1指向A的第一个元素,p2指向B的第一个元素
while p1和p2都还指向一个元素:
    IF A[p1] < 2*B[p2]:
        p1++
    ELSE:
        p2++
        m += A中p1到表尾的元素个数

返回 (n,m,L)
```

```
//排序+逆序数统计
Sort-and-Count(L)
IF L只有一个元素 then
    没有逆序
ELSE
    把这个表划分成两半:
        A包含前n/2个元素
        B包含n/2之后的元素
    (nA,mA,A) = Sort-and-Count(A)
    (nB,mB,B) = Sort-and-Count(B)
    (n,m,L) = Merge-and-Count(A,B)

返回(n+nA+nB, m+mA+mB, L) //返回逆序数，夸张逆序数，排好序的表L
```

根据主方法，$f(n) = O(n) = n^{log_2 2}$，因此$T(n) = \Theta(nlogn)$

5.动态规划

```
A[maxn],B[maxm]  //两个原串
OPT[maxn][maxm]  //OPT[i][j]表示第一个串的前i个基因与第二个串的前j个基因的最小代价
//delta为空隙罚分(用字符'-'表示空隙)，alpha[i][j]为第一串的第i个与第二串的第j个的错配罚分
for i in range(n):
    OPT[i][0] = i*delta
for j in range(m):
    OPT[0][j] = j*delta
对于每个OPT[i][j],保存对应的方案，用序列L1[i][j]和序列L2[i][j]进行保存(可以看做两个3维数组)
for i in range(1,n):
    for j in range(1,m):
        OPT[i][j] = INF  //先设为无穷大
        //如果在第一串末尾加个空隙更优
        if OPT[i-1][j] + delta < OPT[i][j]:
            OPT[i][j] = OPT[i-1][j] + delta
```

```
            L1[i][j] = L1[i-1][j].append('-')
            L2[i][j] = L2[i-1][j]
        //如果在第二串末尾加个空隙更优
        if OPT[i][j-1] + delta < OPT[i][j]:
            OPT[i][j] = OPT[i][j-1] + delta
            L1[i][j] = L1[i-1][j]
            L2[i][j] = L2[i-1][j].append('-')
        //如果强行将第一串和第二串末尾基因进行错配更优
        if OPT[i-1][j-1] + alpha[A[i]][B[j]] < OPT[i][j]:
            OPT[i][j] = OPT[i-1][j-1] + alpha[A[i]][B[j]]
            L1[i][j] = L1[i-1][j-1].append(A[i])
            L2[i][j] = L2[i-1][j-1].append(B[j])
```

最后我们只需要将序列L1[n][m]和序列L2[n][m]与题目给定的对比方案进行比较即可。
假如对比结果不同，那么要么题目给定的方案不是最优，要么最优方案不唯一。

6.网络流

- 因为所有节点的入边数等于出边数，所以沿着边每当进入一个点，定能找到另一条边离开该结点进入下一个结点。
- 假设我们从s点开始dfs，每走过一条边就把它删去，那么一定能够走回s点。每次走回s点，走过的路径就构成了一个环，并且我们已经将该环删去。
- 因此每当我们从s走到了t，一定能找到一条路径从t走到s。也就是若存在路径s->t，那么存在环s->t->s
- 又因为存在k条边不相交的s->t路径，所以存在k个边不相交的s->t->s环。
- 所以一定存在k条从t到s的边不相交的路径。 证毕。

7. NP完全问题

- 证书为k个患者的治疗区间。我们可以在多项式时间内判断k个患者的治疗区间是否发生冲突，因此：$TCSP \in NP$
- 现在我们尝试将Independent Set问题规约到TCSP。对于n个患者，我们使用O(n^2)的的时间，来判断患者两两之间是否发生时间冲突。
- 我们可以将患者当做结点，发生时间冲突的患者之间连上一条边。现在我们将TCSP问题看做黑箱，假如通过调用TCSP问题我们能够判断，是否能够无冲突的治疗k个患者。那么是否能无冲突的治疗k个患者可以等价于=>是否存在k个两两之间没有边相连的点集。因此我们就能够得到Independent Set问题的解。因此，$IndependetSet \leq_p TCSP$
- 又因为$IndependentSet \in NPC$，所以$TCSP \in NPC$。证毕。

8.方法不限

贪心法：

```
//定义min[i]为1~i的最小成交价格，p[i]为第i天的成交价格。
int ans = -INF, min[0] = INF //INF表示无穷大(-INF为无穷小)
for(int i=1;i<=n;i++){
    min[i] = min(min[i-1], p[i]);
    ans = max(p[i]-min[i]);
}
//最终答案即为ans，时间复杂度O(n)
```