



4-5 P61题

两两配对

10

b. 策略①若 n 为偶数，则每遇到不匹配结果为“好-好”的时候，随机取其中一个，若为其它情况，则弃置两块芯片。
 要保证让新好的成
 好-好芯片 || ②若 n 为奇数，则遇到“好-好”时，随机取一个，取完后，继续计算取出
 依然组队一半。但如
 是一个好-好，要加
 入原组队那块芯片，否则弃置之。
 证明：当 n 为偶数时 a. 某些做得和的一样，如
 $n=2m$ $r \leq m-1$
 r 个 好-好

$n=2h$, 好-好 好: 至少 $(h+1)$ 报告它好，坏: 也可以有 $(h+1)$ 报告它

$\Rightarrow \frac{n-r-h}{2} - \frac{r-h}{2} = \frac{n-2r}{2}$, 由 $r \leq m-1$, 故 $\frac{n-2r}{2} \geq 1$, 即
 取出的好芯片至少比坏芯片多一块。

$\left\{ \begin{array}{l} \text{坏-坏} \\ \frac{r-h}{2} \text{ 坏} \end{array} \right.$

$\left\{ \begin{array}{l} \text{好-好} \\ \frac{n-r-h}{2} \text{ 好} \end{array} \right.$

(2) 当 n 为奇数时

$n=2m+1$ $r \leq m$

① 取出的芯片为好

$$\frac{n-r-1-h}{2} - \frac{r-h}{2} = \frac{n-2r-1}{2} \geq 0$$

r 个 $(n-r-1)$ 好

\Rightarrow ① 若 $\frac{n-r-1-h}{2} + \frac{r-h}{2}$ 为偶数，加入芯片可使好-坏；
 ② 若为奇数，则 $\frac{n-2r-1}{2} \geq 1$, 不需加入芯片可使
 好-坏。

$\left\{ \begin{array}{l} \text{坏-坏} \\ \frac{r-h}{2} \text{ 坏} \end{array} \right.$

$\left\{ \begin{array}{l} \text{好-好} \\ \frac{n-r-1-h}{2} \text{ 好} \end{array} \right.$

② 取出的芯片为坏

$(r-1)$ 坏, $(n-r)$ 好

$\Rightarrow \frac{n-r-h}{2} - \frac{r-1-h}{2} = \frac{n-2r+1}{2} \geq 1$. $\left\{ \begin{array}{l} \text{① 若 } \frac{n-r-h}{2} + \frac{r-1-h}{2} \text{ 为奇数，不需加入芯片可使好-坏。} \\ \text{② 若 } \frac{n-r-h}{2} + \frac{r-1-h}{2} \text{ 为偶数，即 } \frac{n-2r+1}{2} \geq 2 \text{ 时，} \end{array} \right.$

9. Hankou Road Nanjing China

$\frac{n-2r+1}{2} \geq 2$ 加入一块 cable: 0909

12# HLR 71#

2. 设第一个应聘者为 h , 则小孩被雇用两次
满足: 所有 $n-h$ 个比 h 大的元素中, h 需
排在最前面, 因为这 $(n-h)$ 个数中, h 可能排在

$1 \sim n-h$ 任意一个位置, 故概率为 $\frac{1}{n-h}$, 而
 h 取 $1 \sim n$, 每个概率为 $\frac{1}{n}$, 故总概率是
 $\frac{1}{n} \times \sum_{i=1}^{n-1} \frac{1}{n-i} = \frac{1}{n} \times (n-1 + n-2 + \dots + 1) \approx \frac{1}{n} \ln n$.

5.4.6.

① 令 X_i 为第 i 个箱子为空, 则 $P(X_i) = \left(\frac{n-1}{n}\right)^n$, 则 $E(X) = \left(\frac{n-1}{n}\right)^n \times n \approx \frac{n}{e}$

② 令 X_i 为第 i 个箱子刚好有 1 个球, 则 $P(X_i) = \binom{n-1}{n-1} \times \left(1 - \frac{1}{n}\right)^{n-1} \times \left(\frac{1}{n}\right)^1$ (n 球

伯努利试验) $= \left(1 - \frac{1}{n}\right)^{n-1} \approx \frac{1}{e}$, 故 $E(X) \approx \frac{n}{e}$

6.5.9. 首先将 h 个链表的第一个结点组织成最小堆, 共 $O(h)$,
然后, 移去堆顶最小元素, 再把插进待元素放在链表的下一个元素
这样, 每次执行 n 步, 每次插入的复杂度为 $O(1)$, 故
共 $O(n^2)$.

6-3. a.

2	5	12	14
4	8	16	∞
5	9	∞	∞
∞	∞	∞	∞

b. 显然, $A[1][1]$ 为最小元素, 去除之后, 需
比较其左边和下边的两个数据, 再把小的
那个移到原位置。此时, 问题变为对
 $(m-1) \times n$ 或 $m \times (n-1)$ 矩阵去除最小元素, 依
此递归, 直到碰到矩阵的边界时为止
> 把矩阵右角元素移到该位置, 然后,
终止算法。复杂度: $T(p) = T(p-1) + O(1)$

3	8	12
4	9	16
5	0	18

逆序

c. 把元素放在原矩阵最后一步 $\xrightarrow{O(m+n)}$
元素后面, 然后跟 c 相似的方法向上递归。
(每次跟较大的那个元素交换)

3	8	12
4	9	16
5	0	∞

3	8	12
4	9	16
5	0	18

3	8	12
4	②	16
5	9	18

3	2	12
4	8	16
5	9	18

3	1	12
4	8	16
5	9	18



e. 依次插入 n^2 个元素到空的矩阵中，每次插入为 $O(n+m)=O(n)$ ，
故共 $O(n^3)$ ，然后，依次把最小元素移出，每次移出为 $O(n+m)=O(n^2)$
共 n^2 个，故共 $O(n^3)$

f. 考虑左下角的元素，把待定元素与该元素比较，若小则向上
走，否则向右走，复杂度为 $O(m+n)$ 。

7.2-4 对于插入排序而言，其比较次数数为 $\Theta(n+d)$ ，其中 d 为逆序对的个数

①对于插入排序而言，其比较次数数为 $\Theta(n+d)$ ，其中 d 为逆序对的个数

因此，当逆序对很少时，插入排序将接近 $O(n)$ 级别

②对于快速排序而言，对于作为标志的元素而言，若前边一半比它小，
一半比它大，即前面有一半元素与之形成逆序对，其平均性能最好，而
若前边没有和它形成逆序对的，即都比它小，将其划分将出
现一个空，这是极其糟糕的。同样，若前边和它形成逆序对的元素
太少，其划分同样很糟糕。而对于一个几乎有序的划分，出现这种精
糕的划分概率相当大，故最终将接近 $O(n^2)$ 级别。

7.4-2. $T(n) = \min_{0 \leq x \leq n-1} (T(x) + T(n-x-1) + \Theta(n))$, 设 $T(n) \geq cn \lg n$.

$$T(n) = \min_{0 \leq x \leq n-1} (c n \lg x + c(n-x-1) \lg(n-x-1) + \Theta(n))$$

$$= \min_{0 \leq x \leq n-1} (c \frac{n-1}{2} \lg \frac{n-1}{2} + c(\frac{n-1}{2}) \lg(\frac{n-1}{2}) + \Theta(n))$$

$$= c(\frac{n-1}{2}) \lg(\frac{n-1}{2}) - c(n-1) + \Theta(n)$$

$$= c(n-1) \lg(n-1) - cn + c + \Theta(n)$$

$$\geq cn \lg \frac{n}{2} - cn + c - c \lg(n-1) + \Theta(n)$$

$$\geq cn \lg \frac{n}{2} - cn + c$$

当 x 取多少时， $T(n) \geq cn \lg n$ 。

注： $c n \lg x + c(n-x-1) \lg(n-x-1)$ 在 $(0, \frac{n-1}{2})$ 和 $(\frac{n-1}{2}, n-1)$ 取最小值（两次求导）。

$$\begin{aligned}
 1-4. & \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{h=1}^{n-i+1} \frac{2}{h} \\
 & \geq \sum_{i=1}^{n-1} \sum_{h=1}^{n-i+1} \frac{2}{2h} \\
 & \geq \sum_{i=1}^{\frac{n}{2}} \sum_{h=1}^{\frac{n-i}{2}} \frac{1}{h} \geq \sum_{i=1}^{\frac{n}{2}} \sum_{h=1}^{\frac{n}{2}} \frac{1}{h} \geq \sum_{i=1}^{\frac{n}{2}} n! (g^n) = \Omega(n(gn))
 \end{aligned}$$

$$\begin{aligned}
 & \text{或 } \sum_{i=1}^{n-1} \sum_{h=1}^{n-i+1} \frac{1}{h} \geq \sum_{i=1}^{n-1} \int_1^{n-i+1} \frac{1}{x} dx \\
 & = \sum_{i=1}^{n-1} [n(n-i+1)] = (n \prod_{i=1}^{n-1} (n-i+1)) \\
 & = nn!
 \end{aligned}$$

4-5.

不同子数组元素之间不形成逆序，而 h 个元素的子数组中，
期望的逆序对为 $(h^2 - h) / 2 = \frac{h(h-1)}{4}$ ，插入排序次数为 $O(h^2 + \frac{h(h-1)}{4})$ 。
则 $O(h^2 + \frac{h(h-1)}{4}) \times \frac{n}{h} = O(1 + \frac{h-1}{4}) \times n = O(nh)$ 。
而对快速排序而言，则从 n 划分到 h 为止，递归树高度为 $\lg \frac{n}{h}$ ，每层为 n ，故总为 $O(n(gn/h))$ 。

$$\begin{aligned}
 \text{理论上: } & \exists cgn \ln n \geq c nh \lg n \ln (n/h) \\
 & \Rightarrow gh \geq \frac{c}{c'} k,
 \end{aligned}$$

实际上：通常由实验来确定。

4-6.

假设 $0 < h \leq \frac{n}{2}$ ，则要比 $(1-2)$ 更糟的划分，
因为在 3^n 个数中，至少 2^n 个要出现在最小的那部分（离左最近）。
2 的根元素是 $32^2(1-2)$ ， $3^n : 2^n$ ，总概率为 $2 \times (32^2 - 32^3 + 2^3)^n$

$$2 \times (32^2 - 22^3) = 62^2 - 42^3$$

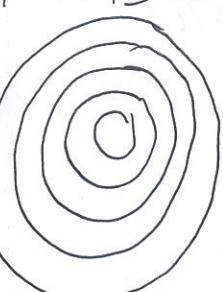
1-4. 每个子数组中有 $h!$ 个排列， $\frac{n}{h}$ 个数组构成的决策树至少有 $(h!)^{\frac{n}{h}}$ 个叶结点，
而对任意一种算法，~~我们~~ 构成的决策树至少有 $(h!)^{\frac{n}{h}}$ 个叶结点。
故有 $2^n \geq (h!)^{\frac{n}{h}} \Rightarrow h \geq \frac{n}{h} \ln(h!) \Rightarrow \frac{n}{h} \times \frac{h}{2} \times \ln(\frac{h}{2}) = \frac{n}{2} \ln(\frac{h}{2}) = \Omega(n \ln h)$

注：若不是简单的计算 $\frac{n}{h} \times \Omega(h \ln h) = \Omega(n \ln h)$ ，不严谨，因为这里说明了任
对每个子数组单独排列算法，其下界为 $n \ln h$ ，但可能有其他算法会
对不同数组间元素进行比较。



4-4. 圆的面积为π，需要把圆平均分为n部分，每部分面积为 $\frac{\pi}{n}$ 。

设有n个圆环，外径半径 a_i ，内径 a_{i-1} ，则有



$$\left\{ \begin{array}{l} \pi \times (a_1^2 - a_0^2) = \frac{\pi}{n} \\ \pi \times (a_2^2 - a_1^2) = \frac{\pi}{n} \\ \vdots \\ \pi \times (a_n^2 - a_{n-1}^2) = \frac{\pi}{n} \end{array} \right.$$

则有 $a_i^2 = \frac{i}{n} \Rightarrow a_i = \sqrt{\frac{i}{n}}$

即分割为 $[0, \sqrt{\frac{1}{n}}), [\sqrt{\frac{1}{n}}, \sqrt{\frac{2}{n}}) \dots [\sqrt{\frac{n-1}{n}}, 1)$

则问题可以转化为前面
不用排序的例子。

注意：

3. 首先，令记录原数组各元素位置的数组B大小为A.length，其中B[i] = B[i] (为了后续比较使用)
 即B[i]是A[i]的正确位置

```

while i < A.length
    if A[i] is correctly placed;
        i++;
    else
        exchange A[i] with A[B[A[i]]].
    
```

~~判断A[i]是否处于正确位置的方法：~~

已知A[i]的当前位置为i，其最后应处于的位置为B[A[i]]，若 $B[A[i]] = i$ [A[i]]，显然正确，但还有一种情况是若 $A[i]$ 已经被放到正确位置了，由于 $B[A[i]]$ 将不会指向其他的A[i]的正确位置。此时再遇到已被放到正确位置的A[i]时，将有 $i > B[A[i]]$ ，则A[i]又将被移至正确判定方法为：

9. HanKou Road Nanjing China

Cable: 0909

if ($i > B[A[i]] \&& i \leq B[A[i]+1]$)

(注意：每次把 $A[i]$ 交换到正确位置后，注意让 $B[A[i]] - 1$ 。

令 $A[3\ 4\ 3\ 4\ 2]$ ，模拟排序过程可以帮助理解)

8-3 解：

a. 算法：(设有 m 个整数，显然 $m \ll n$)

① 计算出各整数的位数， $O(n)$

② 按位数对所有整数分组 $O(m)$ (计数排序)

③ 对每组进行基数排序 $\sum_{i=1}^d (c + m_i) = O(n)$

b. 算法：① 先按首字符对所有字符串进行计数排序，
并把首字符相同的归于一组
(计数排序)

② 逐一地把每一组的字符串进行比较 (每个字符串末尾补一个'0'进行比较)
(计数排序)

遇到'0'时

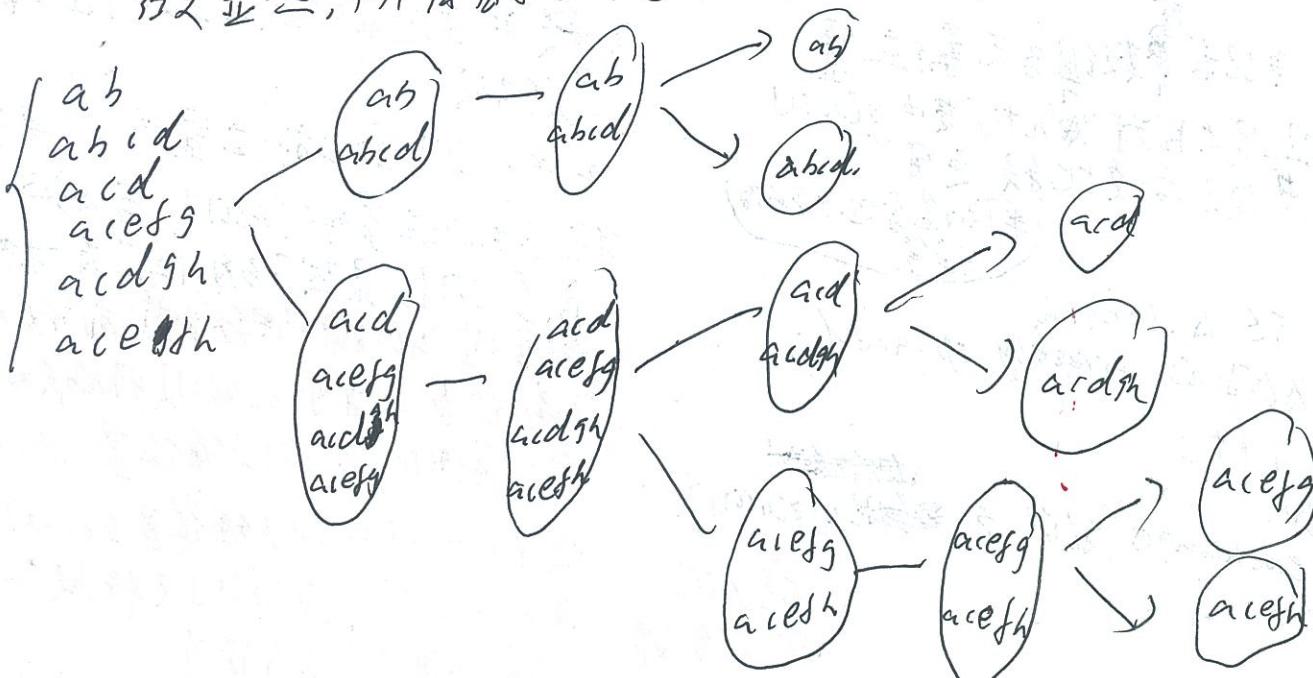
分析：显然，每个字符串最多参与比较的次数为 length/l ，

即所有字符串参与的总比较次数为 $O(n \cdot m)$ 。

而对于每次比较，设有 h 个字符串比较来毛时

$O(h)$ ，而 h 个字符串要贡献了多少比较次数中的 h 次。

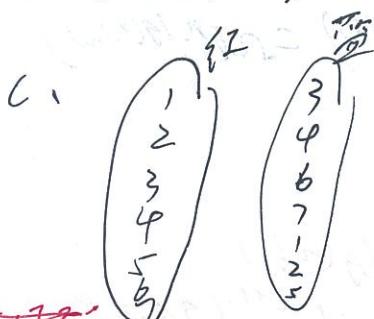
故显然，所有的耗时为 $O(n \cdot m) = O(n)$





8-4 题：

- b. 第一个红色水壶可能有几种而已对，第二个 $(n-1)$ 种 —，第 $n-1$ 种，
 共 $n!$ 种，每次比较有 3 种可能： $<$, $=$, $>$, 故可构造一棵三叉树。
 满足 $3^h \geq n!$ $\Rightarrow h \geq \lceil \log_3 n! \rceil \Rightarrow h \in \Omega(n^{1+\epsilon})$



随机取从红中随机取一个水壶，作为 pivot，然后依次快速排序，依次与所有蓝色水壶比较，保存与该水壶相等的蓝色水壶，然后把蓝色水壶分为

\langle = \rangle ，再以 \langle 为 pivot，依次

与红色水壶比较，划分 \langle = \rangle 。

然后递归处理 \langle 与 \rangle ， \langle 与 \rangle 。

算法分析，与 7.4 节快速排序的随机化平均类似

$$\frac{A[1] + A[2] + \dots + A[h]}{h} \leq \frac{A[2] + A[3] + \dots + A[h+1]}{h} \Leftrightarrow A[1] \leq A[h+1]$$

$$\frac{A[2] + A[3] + \dots + A[h+1]}{h} \leq \frac{A[3] + A[4] + \dots + A[h+2]}{h} \Leftrightarrow A[2] \leq A[h+2]$$

$$\frac{A[n-h] + A[n-h+1] + \dots + A[n-1]}{h} \leq \frac{A[n-h+1] + \dots + A[n]}{h} \Leftrightarrow A[n-h] \leq A[n]$$

成立。

d. 由上可知排序只需满足：共 n 个数组，每组有 n/h 个元素，对每组元素进行归并排序，复杂度为 $O(n/h \lg n/h)$ ，共 n 组，故有 $n \times O(n/h \lg n/h) = O(n \lg (n/h))$

9. Hankou Road Nanjing China

Cable: 0909

$$A[1] \leq A[h+1] \leq \dots$$

$$A[2] \leq A[h+2] \leq \dots$$

$$A[n-h] \leq A[h+n] \leq \dots$$

c. 首先把 h 个子数组中的最小元素取出来，构建一个最小堆，复杂度为 $O(h)$
 然后，移出最小元素，并取该元素所在元素插入堆中，共需 $O(1/h)$ ，共
 移出 n 个元素，故总时间为 $O(h + n(1/h)) = O(n(1/h))$

f. 每个子数组共有 $(n/h)!$ 个排列， h 个子数组共 $[(n/h)!]^h$ ，破坏构造策略
 有 $h! \geq [(n/h)!]^h \Rightarrow h! \geq h!(n/h)! = h \times \prod_{i=1}^{n/h} (i/n)$ $\geq \prod_{i=1}^{n/h} (i(n/h)) = n^{n/h}$

8-6. 解：

$$a_b(2^n) = \frac{2^{2^n}}{\sqrt{\pi n}} (1 + O(\frac{1}{n})) \quad (\text{练习 C.1-13})$$

故有 $2^n \geq \frac{2^{2^n}}{\sqrt{\pi n}} (1 + O(\frac{1}{n}))$ 或 $= g(2^n) - 2\ln(n!)$
 $\Rightarrow \int_1^{2^n} g(x) dx - 2 \int_1^{n+1} g(x) dx$
 $\Rightarrow h \geq \ln \left[\frac{2^{2^n}}{\sqrt{\pi n}} (1 + O(\frac{1}{n})) \right] = \ln 2^{2^n} - \ln \sqrt{\pi n} + O(1) + O(\frac{1}{n})$
 $= 2n - O(n)$

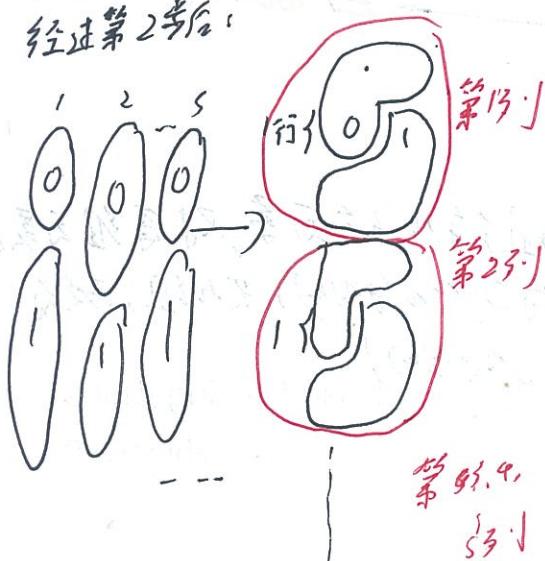
c. 对任意相邻元素 a, b ，其他元素要么比它们都小，要么比它们都大，
 无法分辨 a, b 究竟谁排前面，故 a, b 必须比较。

d. 考虑 $\langle a_1, b_1, a_2, b_2, \dots, a_n, b_n \rangle$ 右分佈的子数组

$\langle a_1, a_2, \dots, a_n \rangle$ 由于 $(a_1, b_1), (b_1, a_2), (a_2, b_2), (b_2, a_3), \dots, (b_{n-1}, a_n), (a_n, b_n)$ 两两必须比较，故至少
 比较 $2n-1$ 次。

8-7 角度：

d. 经过第 2 步后：



由图可知，每列折叠后至少出行一列行进，
 最多 s 行。



- e. 显然, 第6步后, 最多 $s \times s = s^2$ 个元素组成矩阵区域
- f. 但由于 $\sqrt{2}s^2 < 2s^2$, 故 $s^2 < \frac{2}{\sqrt{2}}$, 即 第5步后那 $\frac{s^2}{2}$ 个元素组成的区域现在是第2步的一半。
- ①若这些元素处于同一行, 由于这些元素之前为全0, 之后为全1, 在同一列经过排序后, 则整个数组已经列为优先有序。
- ②若这些元素不处于同一行, 则其一定有一部分在某一行, 另一部分在另一行。只经过第6步后, 其移到同一行。再经过排序后, 将像①样形成行列优先有序。

综上, 得证。

9. 第1行 第2行

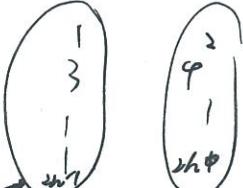
由图可知, 每一列内部最多有一行矩阵行, 相邻两列最多也形成一行矩阵行, 故最多 $s + (s-1) = 2s-1$ 行。



第1步：两两比较每对元素，对较小者再两两比较，直到剩下1个元素为止，该元素为最小值。而第二小元素又可能与前一个最小值比较，故要在最小值比较过的元素中，找出这些元素的 ~~最小值~~，即为所求。

分析：构建一棵二叉树，如图所示，易知叶结点个数即为n，内结点个数即为比较的次数，易知此二叉树内结点比叶结点少1（设内结点n，则 $h+n = 2^{h+1} \Rightarrow n = h+1$ ），故其比较 $n-1$ 次（也由于每个非最小元素恰好输出一次得）。则最小元素比较通过的次数为树的高度h，易知 $h \leq n \leq 2^{h+1}-1$ ，且 $h+1 \leq \lceil \lg n \rceil + 1 < h+2$ ，故 $h = \lceil \lg n \rceil + 1$ ，则最多还需比较 $\lceil \lg n \rceil + 1 - 2 = \lceil \lg n \rceil - 1$ 。综上共比较 $n + \lceil \lg n \rceil - 2$ 次。

9.1-2. 先证一次比较最多能排除一个元素不为最小和一个元素不为最大的能力，若n个元素分成一对一对比较，则每次比较均能排除一个元素不为最小和一个元素不为最大。而若任取两次比较，其中必有某元素一定会比较 (a, b), (c, d)，不妨设 $a < b, c < d$ ，则能排除 a 不为最大， c 不为最小，其剩余的排除元素不为最小和最大的次数是3，故得证。我们先对n个元素（不妨先设为偶）来进行比较，得分为三部分：



其中左边不可能有
最大，右边不可能有

考虑我们若跨集合比较，要成功，就任意两个元素，同上边两次比较果元素重合的分析，至若 $a_1 < a_2$ ，则必然 a_1 是最小的，故能减少一种可能，进而我们要排除 $n-1$ 个元素为最小的情况下，发生矛盾， $n-1$ 个元素为最大的可能，是需 2^{n-2} 种，至少需



$\frac{n}{2} \times 2 + [2(n-2) - \frac{n-1}{2} \times 2] \times 1 + \frac{n}{2} \times 1 = \frac{3n}{2} - 2$ 次比较
 逐对比较经中五零比较的次数。
 对于 $n=2h+1$ 情况多出来的一个元素可以和任一分组中第三元素比较，是该情况下操作八分之一次。
 当 n 为奇数时， $[2n-2 - \frac{n-1}{2} \times 2] \times 1 + \frac{n-1}{2} \times 1 = \frac{3n}{2} - 3$ 次比较

- ① 当 $n=2h$ 时， $\frac{3n}{2} - 2 = (\frac{3n}{2} - 2)$
- ② 当 $n=2h+1$ 时， $\frac{3}{2}n - \frac{3}{2} = 3h = (\frac{3n}{2} - 2)$ ，故原题已得证
 (最优算法即为前边书上采用的算法)

对于比小的元素小，则同以滑到那元素，然后加进之)

9.3-3. 模仿前边的划分，则 ~~选择部分后较少的一部分置中位数~~ 每次都调用 select 找出中位数，
~~(会大大增加常数因子，故实际不使用)~~
 时约为 $T(n) \leq 2T(\frac{n}{2}) + O(n)$

9.3-6. (1) 当 n 为偶数时 直接返回
 (2) 当 n 为偶数时，找出中位数，然后对左半部分 和右半部分递归调用。
 (3) 当 n 为奇数时，找出第 $\lceil \frac{n}{2} \rceil$ 和 $\lfloor \frac{n}{2} \rfloor$ 大的元素，然后对左半部分 和右半部分递归调用 (此时 n 变为 $\lfloor \frac{n}{2} \rfloor$)
 复杂度： $T(n, h) = 2T(\lfloor \frac{n}{2} \rfloor, \frac{h}{2}) + O(n) \in \Theta(n \lg h)$

9.3-7. (1) 找出中位数， $O(n)$
 (2) 找出其他每个数与中位数的差， $O(n)$
 (3) 找出这些距离中第 h 小的距离， $O(n)$
 (4) 选出那个距离 $\leq h$ 的元素即 L

该算法的问题是，如果有多个刚好等于 L 的数，则可能添加多个。
 解决该问题的方法：①扫描原数组两次，第一次添加 L 的元素，第二次添加 $= L$ 的元素，直到满足为止。

② 在计算第 h 小的距离时，其方法是：
 同时交换原数组元素
 (可以设计自己的 swap 函数来同时交换两个数组的元素)

9.3-8.

算法：(1) 先找出两个数组的中位数 m_a 和 m_b
(2) 比较 $m_a = m_b$

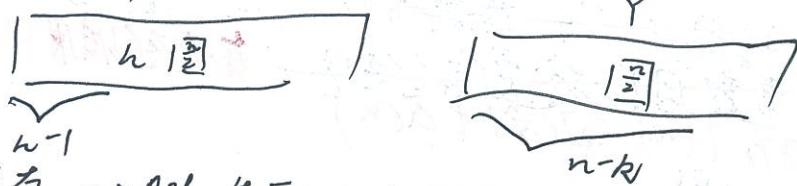
(3) 若 $m_a < m_b$, 则选取 X 中的 ~~较小的~~ 较大的 $\lceil \frac{n}{2} \rceil$ 个元素, Y 中较小的 $\lceil \frac{n}{2} \rceil$ 个元素
若 $m_a > m_b$, 则选取 X 中的 ~~较小的~~ 较大的 $\lceil \frac{n}{2} \rceil$ 个元素, Y 中较大的 $\lceil \frac{n}{2} \rceil$ 个元素
(4) 对新的两个 $\lceil \frac{n}{2} \rceil$ 个元素的数组, 递归调用。

分析：不妨设所求中位数 m 位于 X 中第 k 个元素，则

若 $k < \frac{n}{2}$, 则 X 中有 $k+1$ 个

X 中有 $k+1$ 个元素比 m 小, Y 中有 $n-k$ 个元素比 m 小,

若 $k > \frac{n}{2}$, 则

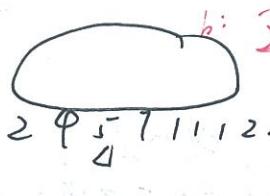
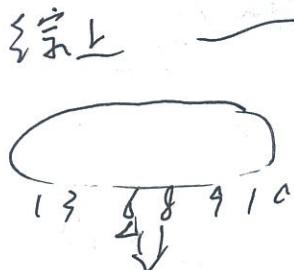


$X > k, Y < k$, 故有 $X > Y$, 选取 X 中较小的 $\lceil \frac{n}{2} \rceil$ 个元素, Y 中较大的 $\lceil \frac{n}{2} \rceil$ 个元素后, 这 $\lceil \frac{n}{2} \rceil + \lceil \frac{n}{2} \rceil$ 个元素中, 比 m 小的有 $(k-1) + (n-k-\lceil \frac{n}{2} \rceil)$
[分别为 $\lceil \frac{n}{2} \rceil - 1$, $\lceil \frac{n}{2} \rceil + 1 - n$]
 $= n-1 - \lceil \frac{n}{2} \rceil = \lceil \frac{n}{2} \rceil - 1$, 故得证。

新的两个数组的中位数。

同理, 若 $k > \frac{n}{2}$, 亦成立。

需要分 n 为奇偶数讨论。



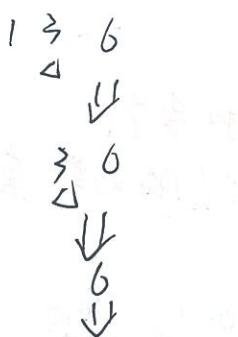
a: 1 3 4 5 7

若 $ma < mb$, 则 $\lceil \frac{n}{2} \rceil$ 的数至少有 $\lceil \frac{n}{2} \rceil + 1 + \lceil \frac{n}{2} \rceil + 1 = (n+1)/2$ 个。

b: 2 6 8 9 10

故 $mb < ma$ 的数必至少有 mb 个。由于它们不能是相同的数，而 mb 本身是不可能的。另外， mb 至少有 $\lceil \frac{n}{2} \rceil + 1 + 1 = (n+1)/2$ 个，故 mb 不能是中位数。

综上



7 11 12. 1 2 4 6

若 $ma < mb$, 则 mb 至少有 $\lceil \frac{n}{2} \rceil + 1$; $\lceil \frac{n}{2} \rceil + 1 = n+2$ 个，故 mb 在群部

7 11 4 11 7

不可能，因 ma 本身也不可能 (至少有 $n+1$ 个)。而 mb 至少有 $\lceil \frac{n}{2} \rceil + 1 = n+1$ 个，故 $mb > ma$ 的部分不可能，而 mb 本身可能是

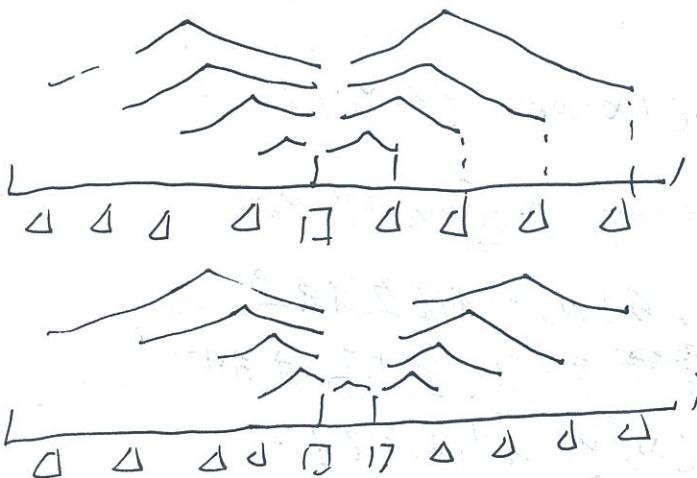
6



3-1. 考虑这 n 口油田的产量

- (1) 若 n 为奇数, 则找出其中位数 m , 让主干线水平穿过该油田即 \rightarrow
(2) 若 n 为偶数, 找出上、下两个中位数, 主干线穿穿过中间任一部分均小

析:



考虑主干线往左(或右)偏移一些
总距离显然增加

3-2.

- (1) 首先对 n 个数按大小排序, $O(n \lg n)$

(2) 从左向右依次累积每个数的权, 遇到第一个累积后权 ≥ 0.5 的数, 即为带权中位数。

- (1) 先对 n 个数求出其中位数 $O(n)$

(2) 利用该中位数将 n 个数进行划分, $O(n)$

(3) 累加中位数左边所有尚数的权重, 若分别 $< \frac{1}{2}$ 和 $\leq \frac{1}{2}$, 则该

中位数即为所求

(4) 否则, 带权中位数处在 $\frac{1}{2}$ 的部分中, 把 $\frac{1}{2}$ 的部分和 $\frac{1}{2}$ 的部分归调用

的权重之和放到 x_n 上, 对 $\frac{1}{2}$ 的部分和 x_n , 重归调用

该算法。

$$\text{复杂度分析: } T(n) = T\left(\frac{n}{2} + 1\right) + O(n) \in \Theta(n)$$

1.  此时每个点的值 x_i 即为坐标

假设带权中位数的权重为 w_s , 其左边的总权重为 w' , 右边为 w'' , 若 x 向左边稍微移动 Δ , 则不难设为向右移动, 则:

$$\sum_{i=1}^n w_i \cdot d(x, p_i) \text{ 的变化为 } w_s \cdot \Delta - w' \cdot \Delta + w'' \cdot \Delta = (w + w_s - w') \cdot \Delta \geq 0.$$

故为最小化

$$3. \text{ 要求 } \sum_{i=1}^n w_i \cdot d(x, p_i) = \sum_{i=1}^n w_i (|x - x_i| + |y - y_i|)$$
$$= \sum_{i=1}^n w_i (|x - x_i| + \sum_{j=1}^n w_j |y - y_j|)$$

由于 x 不一定是输入点中的一个, 故 x, y 相互独立,
我们可以对所有点的横坐标和纵坐标分别
求其带权中位数 x, y , 则 (x, y) 即为此点。

1-4. 角度:

a. 特情况讨论:

当 $i < j < h$ 时,

若取到 $[i, j-1]$ 中的任一元素, 则 (i, j) 不会被比较

取到 i 和 j 比较,

$(i+1, j-1)$ 不比较,
故此时比较的概率 $P = \frac{2}{j-i+1}$

当 $i \leq h \leq j$ 时

i 和 j 比较,

$(i+1, j-1)$ 不比较

$P = \frac{2}{j-i+1}$

3) 当 $i < j < h$ 时, $P = \frac{2}{h-i+1}$

$$4). E[X_{ijh}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ijh}$$

b. $E[X_{ih}]$

$$\begin{aligned}
 &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ijh} = \sum_{i=1}^h \sum_{j=i+1}^n X_{ijh} + \sum_{i=h+1}^{n-1} \sum_{j=i+1}^n X_{ijh} \\
 &= \sum_{i=1}^h \sum_{j=i+1}^n X_{ijh} + \sum_{i=1}^h \sum_{j=h}^n X_{ijh} + \sum_{i=h+1}^{n-1} \sum_{j=i+1}^n X_{ijh} \\
 &= \sum_{i=1}^{h-2} \sum_{j=i+1}^{h-1} \frac{1}{h-i+1} + \sum_{i=1}^h \sum_{j=h}^{i-1} \frac{1}{i-i+1} + \sum_{i=h+1}^{n-1} \sum_{j=i+1}^{i-h+1} \frac{1}{i-h+1} \\
 &= 2 \left(\sum_{i=1}^{h-2} \frac{h-i-1}{h-i+1} + \sum_{i=1}^h \sum_{j=h}^{i-1} \frac{1}{i-i+1} \right) + \sum_{j=h+1}^n \sum_{i=j+1}^{j-h+1} \frac{1}{j-h+1} \\
 &= \sum_{j=h+1}^n \frac{j-h-1}{j-h+1}
 \end{aligned}$$

$$c. \sum_{i=1}^{h-2} \frac{h-i-1}{h-i+1} + \sum_{j=h+1}^n \frac{j-h-1}{j-h+1} \leq (h-2) \times 1 + (n-h) \times 1 = n-2 \leq n$$

$$\text{而 } \sum_{i=1}^h \sum_{j=h}^{i-1} \frac{1}{i-i+1} = (\frac{1}{1} + \frac{1}{2} + \dots + \frac{1}{n-h+1})$$

$$\text{而 } \sum_{i=1}^h \sum_{j=h}^{i-1} \frac{1}{i-i+1} = (\cancel{\frac{1}{2}} + \cancel{\frac{1}{3}} + \dots + \cancel{\frac{1}{n-h+2}})$$

{}

$$\begin{aligned}
 &\quad + (\frac{1}{h} + \dots + \frac{1}{n}) \\
 &= \frac{1}{1} \times 1 + \frac{1}{2} \times 2 + \dots + \frac{1}{n-h+1} \times (n-h+1) + \frac{1}{n-h+2} \times (n-h) + \\
 &\quad + \frac{1}{n-h+3} \times (n-h-1) + \dots + \frac{1}{n} \times 1 \\
 &\leq \frac{1}{1} \times 1 + \frac{1}{2} \times 2 + \dots + \frac{1}{n} \times n = n.
 \end{aligned}$$

故 $E[X_h] \leq \varphi n$.

$$\begin{aligned}
 &\frac{1}{n-2} + \left(\frac{1}{h-1} + \frac{1}{h} \right) + \left(\frac{1}{h} + \frac{1}{h+1} \right) + \dots + \left(\frac{1}{n-h+1} + \frac{1}{n} \right) \\
 &\quad + \left(\frac{1}{n-h+2} + \frac{1}{n-h+3} \right) + \dots + \left(\frac{1}{n-1} + \frac{1}{n} \right) \\
 &\quad + \frac{1}{n} \\
 &\quad + \dots + \frac{1}{n}
 \end{aligned}$$

观察后，知其最繁行。
 而对之，其也最繁。
 而对之，盖归来观察。
 放对往常，才是设计。
 同上知下1的项数为1。

10.1-7. 解：

(1) 设两个队列 A, B，初始设 A 的状态为 active, B 为 inactive

(2) 插入结点时，从 active 的队列进入

(3) 出栈时，先把 active 的队列中除了最后一个元素以外的其它元素入 inactive 队列，再把 pop 最后一个元素，并且，交换两个队列的状态

复杂度：(1) $O(n)$, $O(1)$

(2) 出栈, $O(1)$

A, B
（3. 出栈时的状态应该也可以直接由 A, B 是否空来判断）

10.2-8. 解：

$$x.\text{np} = x.\text{next} \text{ XOR } x.\text{prev};$$


 $\text{head}.\text{np} = \text{null} \text{ XOR } \text{head}.\text{next}$
插入时，令 prev 初始化为 NULL；
则 第二结点： $\text{current} = \text{current} \text{ XOR } \text{prev}$;

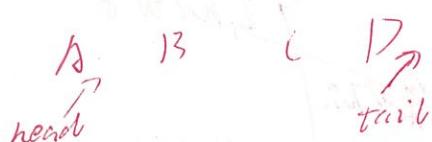
$$\begin{aligned} &\cancel{\text{prev}} = \cancel{\text{current}}; \\ &\text{node} = \text{next}; \\ &\text{current} \end{aligned}$$

插入时, $\text{newNode} \rightarrow \text{np} = \text{tail} \text{ XOR } \text{NULL}$;

$$\begin{aligned} &\text{tail} \rightarrow \text{np} = \text{newNode} \text{ XOR } (\cancel{\text{NULL XOR tail} \rightarrow \text{np}}); \\ &\text{tail} = \text{newNode} \end{aligned}$$

实现链表的过程：

$$\begin{cases} \text{if } \text{head} = \& \text{tail} \\ \text{tail} = \text{head} \end{cases}$$



删除：找到删除结点之后，修改 prev 和 next 的 np.

$$\begin{aligned} \text{prev} \rightarrow \text{np} &= (\text{prev} \rightarrow \text{np} \wedge \text{cur}) \wedge \text{new} \\ \text{next} \rightarrow \text{np} &= (\text{next} \rightarrow \text{np} \wedge \text{cur}) \wedge \text{prev} \end{aligned}$$

由上知，~~search~~ 时，我们由 head 往后得到下一个结点。
而由 tail 往前照相同的过程可以得到前一个结点，故
只要交换 head, tail 即可。



- 10.3-5. ① 先对自由表 L 的各个元素，在 prev 上做坐标，以区别 L 和 F 中的元素。
 ② 令两个指针分别从左右扫描，每次左指针扫描到非空结点为止，右指针扫描到非空结点，交换两个结点的元素，再依次扫描，直到两指针相遇。（复杂度貌似是 $O(n^2)$ ，不清楚）

10.4-5. 每个结点需要有指向父结点的指针。（前序遍历）

- (1) 初始令 $prev = \text{NULL}$; ~~else~~ root 为开始扫描
 (2) 若 $prev = \text{current} \rightarrow \text{parent}$, 才把 current 修正,
 令 $prev = \text{current}$; ~~else~~

current 移向左孩子(空则右孩子, 再空则父结点)

- ③ 若 $prev = \text{current} \rightarrow \text{left}$,
 令 $prev = \text{current}$

current 移向右孩子(空则移向父结点)

11.1-4. 算法：建立两个栈 S 和 S'，每栈中存储着对应元素在数组 T 中的下标，S 存储着对指向对象元素的指针，而 T 则存储着元素在栈中的下标。设

初始化：令 S 和 S' 的 top 指针为 -1。

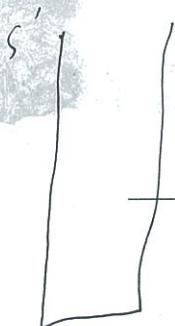
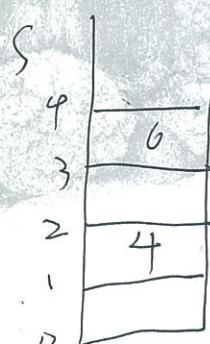
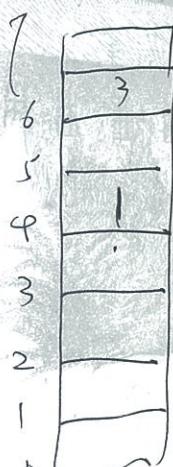
$$S[1[h]] = -1$$

查找：假设要查找的元素对应下标 h，则检测 ~~T[h] == -1~~。

插入：令 $S[++top] = h$, $T[h] = top$,

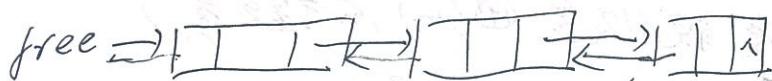
删除：把 S 中栈顶的元素填入待删位置。

$$\begin{aligned} S[1[h]] &= S[top]; & S'[1[h]] &= S'[top]; \\ \rightarrow S[1[h]] &= T[h]; & T[h] &= -1; & top &--; \end{aligned}$$



11.2-1 解：任取两个元素，其在同位置的根是率为 $\frac{1}{n}$ ，总共有 C_n^2 对元素，
故期望值是 $C_n^2 \times \frac{1}{n} = \frac{1}{2} \times n \times (n-1)$

11.2-2 解：① 将所有未占用槽位 ~~链表~~ 用双链表连接起来，free 指向第一个槽位



② 所有处于自由链表的槽，其 flag = 0 不在的，flag = 1

③ 使用了的槽位，其指针指向下一个 ~~映射到该槽位的元素~~ (该元素位于另一个槽位)

④ 插入：

(1) ① 对应槽位为空，存入该元素，flag 置 1，~~next~~ 指针设置为 NULL
并从 free 中链表中删除

(2) ② 对应槽位不为空，检测该位置元素是否映射到该位置
① 是，从自由链表中 ~~用~~ 分配出一个空间（比如第一个元素），令其指针
指向 x 指向的位置， $x \rightarrow next = p$ ；

② 否，说明 x 处于另一个槽位对应的链表中，从自由链表中
分配出一个空间，复制 ~~x~~ 值过去，计算 x 对应的槽位，从
该槽位对应的链表依次检测到 x 的前一个元素，置
前指针为新分配空间的地位。把 ~~x~~ 原来在的槽位
存在新元素

删除：

(1) 若待删除元素所在槽位指空为零，把它加入自由链表即可

(2) 若指针不为空 ① 若待删除元素就在该槽位处，则将其指向的下一元素
复制到该槽位，删除 ~~空~~ 那个元素对应的槽位

② 若该元素在该槽位指向的链表中，把它从该链表
释放即可（前一元素指针指向它的后一元素）

搜索：

(1) 先找到对应槽位，若其元素不为所求，沿其链表寻找。



19

11.4-4. 解：对一个特定的 $h_2(h)$ (假设为 0), 将会产生以下序列

$$\left\{ \begin{array}{l} [h_2(h) \times 0] \bmod m \\ [h_2(h) \times 1] \bmod m \end{array} \right. \quad \text{当 } h_2(h) \text{ 与 } m \text{ 互质时, 但设 } h_2(h) \text{ 为 } 0 \text{ 时}$$

$$[h_2(h) \times i_1] \bmod m = [h_2(h) \times i_2] \bmod m, 0 \leq i_1, i_2 \leq m-1$$

$$\therefore h_2(h) \times (i_2 - i_1) \equiv 0 \pmod{m}$$

$$[h_2(h) \times (m-1)] \bmod m$$

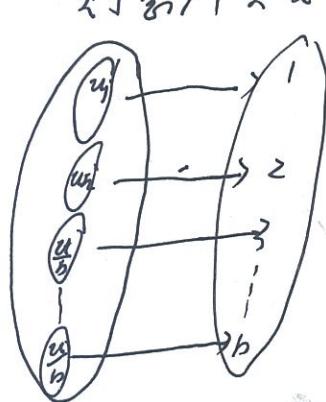
(1) 若 $h_2(h)$ 与 m 互质, 则 $i_2 - i_1 = 0$ 故将遍历所有 m 个位置

(2) 若 $h_2(h)$ 与 m 有最大公约数 d , 假设 $h_2(h) = k, d$,

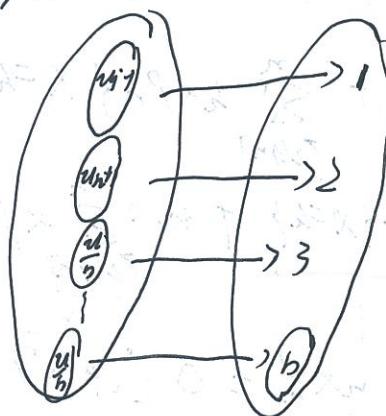
$$m = h_2 d, \text{ 则 } k, d \times (i_2 - i_1) \equiv 0 \pmod{m},$$

令 $i_2 - i_1 = h_2$, 则上等式成立, 即序列每隔 h_2 一
次循环, 未能遍历 $\frac{h_2}{d} = d$ 的位置

11.3-5. 解：先证当集合 U 中映射到 B 中每个元素的数目均相同时（即等），
则命中冲突数最少，如下图。则冲突总数变化为：



→



$$\frac{u_1 \times (u_1-1)}{2} + \frac{u_n \times (u_n-1)}{2}$$

$$\rightarrow \frac{(u_1-1) \times (u_1-2)}{2} + \frac{(u_n+1) \times u_n}{2}$$

$$\text{而 } \frac{u_1 \times (u_1-1)}{2} - \frac{(u_1-1) \times (u_1-2)}{2} = \cancel{u_1-1} < u_n = \frac{(u_n+1) u_n}{2} - \frac{u_n(u_n-1)}{2}$$

故每次往从平衡点往外破坏均衡性，总冲突将增加。

$$\text{故总平均冲突量最小为 } \frac{\frac{n \times (\frac{n}{2}-1) \times b}{2}}{n \times (n-1)} = \frac{n-b}{b(n-1)} > \frac{n-b}{n} = \frac{1}{b} - \frac{1}{n}$$

$$\text{故 } \geq \frac{1}{b} - \frac{1}{n}$$

11.5-1. 解：对 n 用数学归纳法

$$(1) \text{ 当 } n=1 \text{ 时, } P(1, m) = 1 \leq e^{-1 \times 0 / 2m} = 1$$

$$(2) \text{ 当 } n=h \text{ 时有 } P(h, m) \leq e^{-h(h+1)/2m}$$

$$\text{且当 } n=h+1 \text{ 时, } P(h+1, m) \leq P(h, m) \times \frac{m+h}{m}$$

$$\leq e^{-h(h+1)/2m} \times e^{\frac{h}{m}} (1+x \leq e^x)$$
$$= e^{-h(h+1)/2m}$$

11-1解：

$$(1) P = \frac{i-1}{m} \times \frac{i-2}{m} \times \cdots \times \frac{i-h}{m} \leq \left(\frac{n}{m}\right)^n = \left(\frac{1}{2}\right)^n$$

$$(2) 2^{-2\lg n} = 2^{(\lg n)^2} = \frac{1}{n^2}$$

$$(3) \Pr\{X > 2\lg n\} \leq \Pr\{X_1 < 2\lg n\} + \cdots + \Pr\{X_n < 2\lg n\}$$
$$\leq \frac{1}{n^2} \times n = \frac{1}{n}$$

$$(\Phi | E[X] = \sum_{h=1}^n h \times \Pr\{X=h\}$$

$$= \sum_{h=1}^{2\lg n} h \times \Pr\{X=h\} + \sum_{h=1}^n h \times \Pr\{X=h\}$$

$$\leq [2\lg n] \times \sum_{h=1}^{2\lg n} \Pr\{X=h\} + n \times \sum_{h=1}^n \Pr\{X=h\}$$

$$\leq 1$$

$$\leq [2\lg n] \times 1 + n \times \frac{1}{n} = O((\lg n)^2)$$

11-2. 解：a. 易证。

$$b. P_h = \{(\exists i, \text{s.t. } X_i = h) \wedge (\forall i, X_i \leq h)\}$$

$$\subseteq \{ \exists i, \text{s.t. } X_i = h \}$$

$$\leq n \times Q_h.$$

$$c. Q_h = \left(\frac{1}{n}\right)^h \times \frac{(1-\frac{1}{n})^{n-h} \times \frac{n!}{h!(n-h)!}}{n^n}$$

$$< \left(\frac{1}{n}\right)^h \times \frac{n!}{h!(n-h)!} = \frac{1}{h!} \times \frac{n \times (n-1) \times \cdots \times (n-h+1)}{n^n} < \frac{1}{h!} \leq \left(\frac{1}{e}\right)^h$$



d. 令 $e^{h_0}/h_0^{h_0} < \frac{1}{n^3}$

$$\Rightarrow n^3 < \frac{h_0^{h_0}}{e^{h_0}}$$

$$\Rightarrow 3 \lg n < h_0 \lg h_0 - h_0 \lg e$$

~~$$= \lg n + \lg h_0 + \lg h_0 - h_0 \lg h_0 - h_0 \lg e - C$$~~

$$= \frac{\lg n}{\lg \lg n} \times \left((\lg c + \lg \lg n - \lg \lg \lg n - \frac{\lg n}{\lg \lg n} (\lg e)) \right)$$

$$\Rightarrow 3 < \lg c \times \left(\frac{\lg c - \lg e}{\lg \lg n} + 1 - \frac{\lg \lg \lg n}{\lg \lg n} \right)$$

$$\text{令 } x = 1 + \frac{\lg c - \lg e}{\lg \lg n} - \frac{\lg \lg \lg n}{\lg \lg n}, \quad \text{当 } n \rightarrow \infty \text{ 时, } x \rightarrow 1,$$

故存在 n_0 , 使得当 $n \geq n_0$ 时, $x \geq \frac{1}{2}$, 则令 $c = 6$, 有

$n \geq n_0$ 时, (x) 成立。

而当 $0 < h < h_0$, 由于 n 为整数, 故 n 有限, 则 x 只有有限个, 取最大的。使得 $x > \frac{1}{2}$ 且 $3 < n < n_0$ 成立, 则 $c = \max\{6, \lceil x \rceil\}$ 即可。

当 $h \geq h_0$ 时, 令 c 充分大, 使得 $h > 3 > e$, 则有 $Q_h < e^h/h^h = \frac{e^h}{h!}$

$$\leq \left(\frac{e}{h_0}\right)^{h_0} < \frac{1}{n^3}$$

$\frac{c}{h} < 1$ 故成立

$$\begin{aligned} E[M] &= \sum_{h=0}^n h \times \Pr\{M=h\} \\ &= \sum_{h=0}^{h_0} h \times \Pr\{M=h\} + \sum_{h=h_0+1}^n h \times \Pr\{M=h\} \\ &\leq \sum_{h=0}^{h_0} h_0 \times \Pr\{M=h\} + \sum_{h=h_0+1}^n n \times \Pr\{M=h\} \\ &= h_0 \times \sum_{h=0}^{h_0} \Pr\{M=h\} + n \times \sum_{h=h_0+1}^n \Pr\{M=h\}. \end{aligned}$$

$$= h_0 \times \Pr\{M \leq h_0\} + n \times \Pr\{M \geq h_0\} \stackrel{9. Hankou Road Nanjing China}{< \frac{1}{n^2} \times (n-h_0)}$$

$$< h_0 \times 1 + n \times \frac{1}{n} = h_0 = O(\lg^2 \lg \lg n)$$

11-3解：

a. 分别为 $h(h), h(h)+1, h(h)/1+t_2, \dots, h(h)+1+t_{2^k}-1$
故有 $t_1 = \frac{1}{2}, t_2 = \frac{1}{2}, h(h) = (h'(h) + \frac{1}{2}t_1 + \frac{1}{2}t_2) \bmod m$

b. 反证。

若有两个值相等，则有

$$\begin{aligned} h(h) + \frac{1}{2}t_1 + \frac{1}{2}t_2 &\equiv h(h) + \frac{1}{2}t_2 + \frac{1}{2}t_2 \\ \Rightarrow \frac{1}{2}(t_1 - t_2 + t_1 - t_2) &\equiv 0 \pmod{m} \\ \Rightarrow \frac{1}{2}(t_1 - t_2)(t_1 + t_2 + 1) &\equiv 0 \pmod{m} \end{aligned}$$

$$\because m=2^k, \text{ 则有 } (t_1 - t_2)(t_1 + t_2 + 1) = 2^{k+1}$$

由 $(t_1 - t_2)$ 与 $(t_1 + t_2 + 1)$ 奇偶性不同，故要么 $2^{k+1} / t_1 - t_2$ ，要么 $2^{k+1} / t_1 + t_2 + 1$

而 $t_1 - t_2$ 显然 $< m = 2^k$ ，而 $t_1 + t_2 + 1 \leq (m-1) + (m-2) + 1 = 2m-2 \leq 2m = 2^{k+1}$

12.2-6. 解：由于 y 是 x 的后继，且为无右子树，即

x 一定位于 y 的左子树上

(1) 假设 y 不是最底层的其左孩子亦为 x 祖先的祖先，

不妨设 y 的左孩子 y' 亦为 x 祖先的祖先，

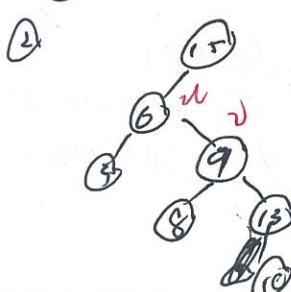
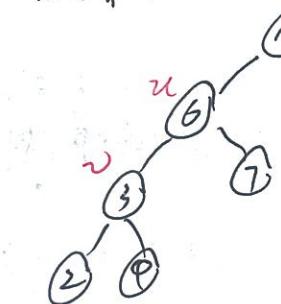
则 $y' < x, y > y'$, 与 y 为 x 的后继矛盾。

12.2-7解：略去，算法运行时间为 $O(n)$

② 算法运行过程中，对每条边至多经过两次，故为 $O(n)$

下证②。假设 $u \rightarrow v$ 为某条边。

① 若 v 为 u 的左孩子，则在访问 u 之前，必先访问 u 的左子树中的最小元素，此时经过 $u \rightarrow v$ 。而在 u 的左子树没有访问完之前， $v \rightarrow u$ 不会经过。当左子树访问完后，通过找最后一个孩子的后继，将经过 $v \rightarrow u$ ，此时，由于 u (无右子树) 的左子树已访问完，将不会再经过 (u, v) ，故最多两次。



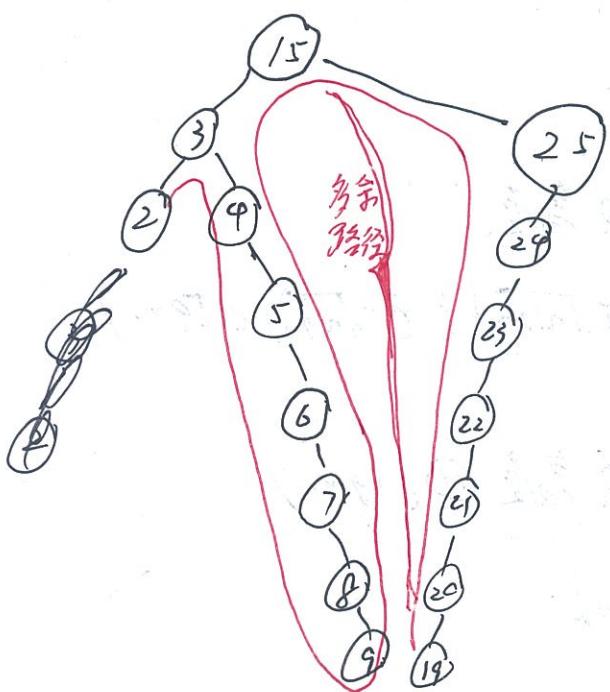
② 若 v 为 u 的右孩子，当访问完 u 后，找 u 的后继将经过 $u \rightarrow v$ ，然后，访问完 u 的右子树后 v 的右子树的最后一个将因为要访问其后继而经过 $v \rightarrow u$ ，故最多两次。



12.2-8 个人认为应该呈 $\Theta(2h^2)$ 或 $\Theta(2h + h) = \Theta(h)$

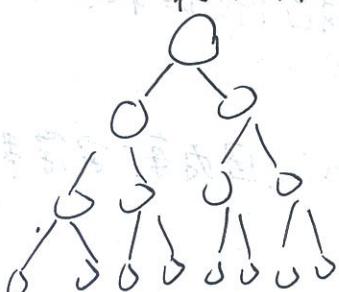
假设访问 2 ~ 19，则

除了中间一段多余的路径之外，其他访问长度在 $2h$ 之间（每条边至多访问 2 次），而多余的路径长度为 $\Theta(h)$ ，故总的时间为 $\Theta(2h + h)$



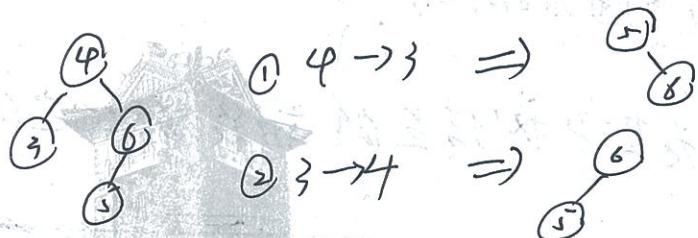
12.3-3. 解：最好情况：树基本平衡，如图，则 T 比较时间为

$$\begin{aligned} T &= 2^1 \times 1 + 2^2 \times 2 + \dots + 2^{h-1} \times h \\ &\Rightarrow T = (h-1) \times 2^{h-1} + 2 \in \Theta(n \lg n) \end{aligned}$$



最坏情况：构成单支树时，比较时间为
 $1 + 2 + \dots + n - 1 \in \Theta(n^2)$

12.3-4. 解：



12.4-2. 解：(1) 先证树叶结点平均深度为 $\Theta(\lg n)$ 的树的平均深度为
边上界为 $O(\sqrt{n \lg n})$ 。

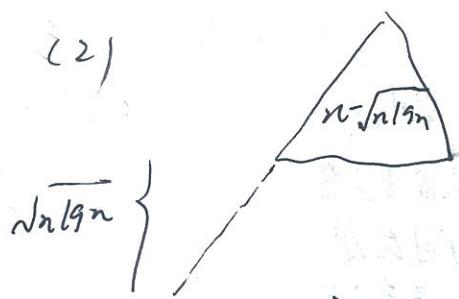
设有一条从根到树最底层的一条路径，其经过的结点高度分别为 $0, 1, 2, \dots, h$ ，则树平均深度 $\geq \frac{1}{n}(0 + 1 + \dots + h) = \frac{1}{n}\Theta(h^2)$

若 n 为 $w(\sqrt{n \lg n})$ ，则 $\frac{1}{n}\Theta(h^2) = w(\lg n)$.

Hankou Road Nanjing China

矛盾。

(2)



显然，该树对高度为 $n(\sqrt{n})$ ，求其平均高度。

平均高度 ~~为上界为~~ $O(\sqrt{n} \times (\sqrt{n} + \sqrt{n}) + (n - \sqrt{n}) \times \lceil \lg n \rceil)$
 $= O(\lg n)$

而 n 结点的树高度下界为 $n(\lg n)$ ，故平均高度 $\Theta(\lg n)$

12.4.3. 解：令 $n=3$ ，有

$\begin{array}{c} 1 \\ \\ 2 \\ \\ 3 \end{array}$	$\begin{array}{c} 1,2,3 \\ 1,3,2 \\ 2,1,3 \\ 2,3,1 \\ 3,2,1 \\ 3,1,2 \end{array}$	对应树如上
--	---	-------

由图可知，每种二叉搜索树并非都能被选取。

② 12-2 解：算法：

(1) 去插入这些字符串，复杂度为 $O(m)$ (因为每个字符串插入到正确的比较为其字符串长度)

(2) 按前序遍历输出， $O(n)$

正确性：

(1) 每个结点为其子树结点的前驱，故地它们先输出

(2) 每个结点上的结点除其父母的
 比较某结点左子树和右子树上结点，
 左子树上结点必小于右子树，故先于它们输出。



12-3. 猜想一：

- (1) 先构建二叉搜索树，
(2) 依此用搜索树的内结点作为 pivot 对快排进行划分。

猜想二：

- (1) 用保持稳定的策略来划分 (比如利用链表)

(2) 递归继续划分

12-4. (1) 一棵 n 结点的二叉树，其左右子树分别可能

对应 $(0, n-1), (1, n-2), \dots, (n-1, 0)$ 。故

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \dots + b_{n-1} b_0$$

$$(2) B(x) = b_0 + b_1 x + \dots + b_n x^n + \dots$$

$$\begin{aligned} B^2(x) &= (b_0 + b_1 x + \dots + b_n x^n)(b_0 + b_1 x + \dots + b_n x^n) \\ &= b_0^2 + (b_0 b_1 + b_1 b_0)x + (b_0 b_2 + b_1 b_1 + b_2 b_0)x^2 + \dots \end{aligned}$$

~~$$x B^2(x) + 1 =$$~~

$$= b_1 + b_2 x + \dots + b_{n+1} x^{n+1}$$

$$x B^2(x) + 1 = 1 + b_1 x + b_2 x^2 + \dots + b_{n+1} x^{n+1}$$

$$= x B(x)$$

$$f(x) = \sqrt{1-4x}, f'(x) = \frac{1}{2} \times (1-4x)^{-\frac{1}{2}} \times (-4), \dots$$

$$f^{(n)}(x) = \frac{1}{2} \times (-\frac{1}{2}) \times (-\frac{3}{2}) \times \dots \times (-\frac{2n-3}{2}) \times (1-4x)^{\frac{1}{2}-n}$$

$$= (-1)^n \times 2^n \times (2n-3)! / (2n-2)!$$

$$\begin{aligned} f(x) &= f(0) + \frac{f'(0)}{1!} x + \frac{f''(0)}{2!} x^2 + \dots + \frac{f^{(n)}(0)}{n!} x^n \\ &= 1 - 2x + \dots + \frac{(-1)^n \times 2^n \times (2n-3)!}{n!} x^n \end{aligned}$$

$$\text{故 } B(x) = \frac{1}{2} (1 - f(x)) = 1 + \dots + \frac{2^n \times (2n-1)!}{(n+1)!} x^n$$

$$\Rightarrow b_n = \frac{2^n \times (2n-1)!}{(n+1)! \cdot 2^n} = \frac{2^n \times (2n-1)!}{(n+1)! \cdot 2^n \cdot (2n-1)!} = \frac{9. HanKou Road Nanjing China}{(n+1)! \times n!} \quad \text{Cable: 0909}$$

$$= \frac{1}{n+1} (2^n)$$

d. 斯特林近似公式 $n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + O(\frac{1}{n}))$

$$\text{则 } \frac{1}{n+1} \binom{2n}{n} = \frac{1}{n+1} < \frac{(2n)!}{n! \cdot n!}$$

$$\begin{aligned} & \frac{\frac{1}{n+1} < \frac{\sqrt{4\pi n} \times \left(\frac{2n}{e}\right)^{2n}}{(2\pi n) \times \left(\frac{n}{e}\right)^{2n}}}{=} = \frac{1}{n+1} \times \frac{\sqrt{4\pi n}}{2\pi n} \times e^{4n} \\ & = \varphi^n \times \frac{1}{\sqrt{\pi} \times \sqrt{n} \times n+1} = \varphi^n \times \frac{1}{\sqrt{\pi} n^{\frac{3}{2}}} \times \frac{1}{1 + \frac{1}{n}} \\ & \in \frac{\varphi^n}{\sqrt{\pi} n^{\frac{3}{2}}} (1 + O(\frac{1}{n})) \end{aligned}$$

13.2-5. 解：可以 $O(n)$ 次 把 $\sqrt[3]{n}$ 的根号移到跟号一样，
此时，问题可以递归到两棵树上，设有
 $T(n) = T(n_L) + T(n_R) + O(n)$, $n_L + n_R + 1 = n$,
 $\text{则 } T(n) \in O(n^2)$

13.3-5. 当只有两个结点时，必为 1 黑 1 红

(3) 当结点数 ≥ 3 时，若出现需要调整的情况，
则树中必至少有 2 个红结点，而调整过程中，
最多减少一个红结点，故调整后，必
至少还存在 1 个红结点。下次出现需要调
整的情况下，树中必至少存在 2 个红结点。

如此循环。

13.4-1 解：程序执行共有四种情况。

(1) 情况 1 中若 B 为根，则处理后 D 变为根，且为黑色，进入
情况 2 或 3。
~~情况 2 或 3~~

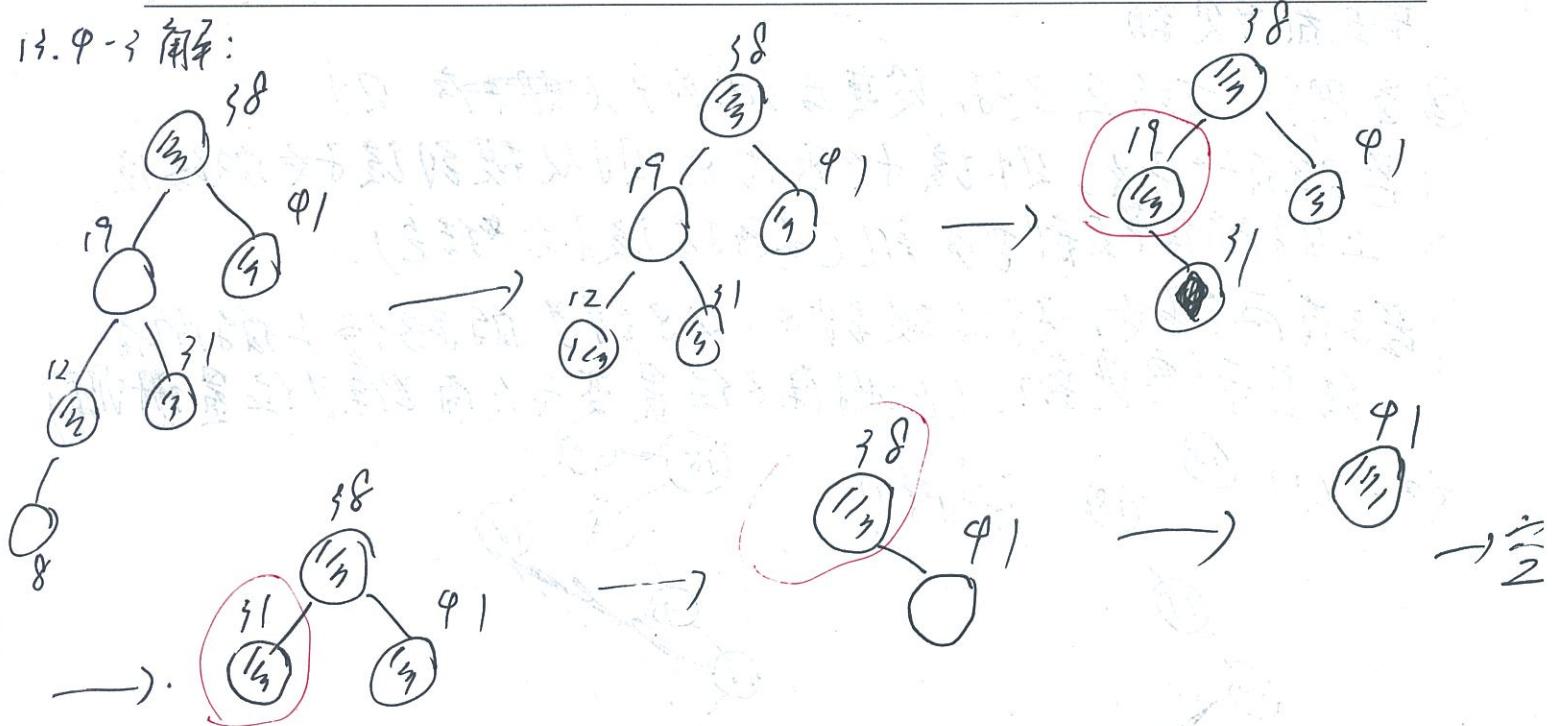
(2) 情况 2 中若 B 为根，则 B 为黑，处理后将跳到循环且
B 依旧为黑；

(3) 情况 3 中 B 不变，进入情况 4

(4) 情况 4 中若 B 为根，则处理后 D 为根，其是 由 B 的
颜色决定的，也为黑 ($D = l.root \neq x, color = black$)
不会将其变



13.4-3 解:

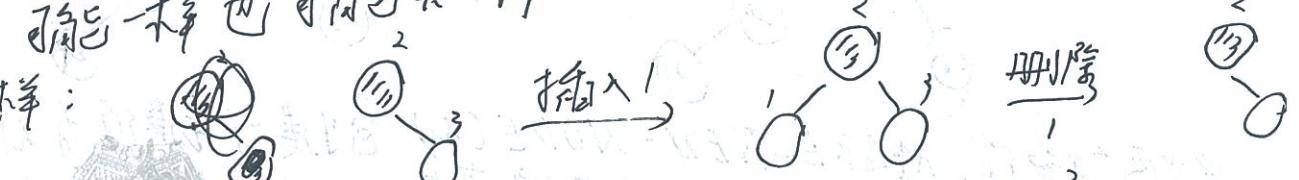


13.4-4.

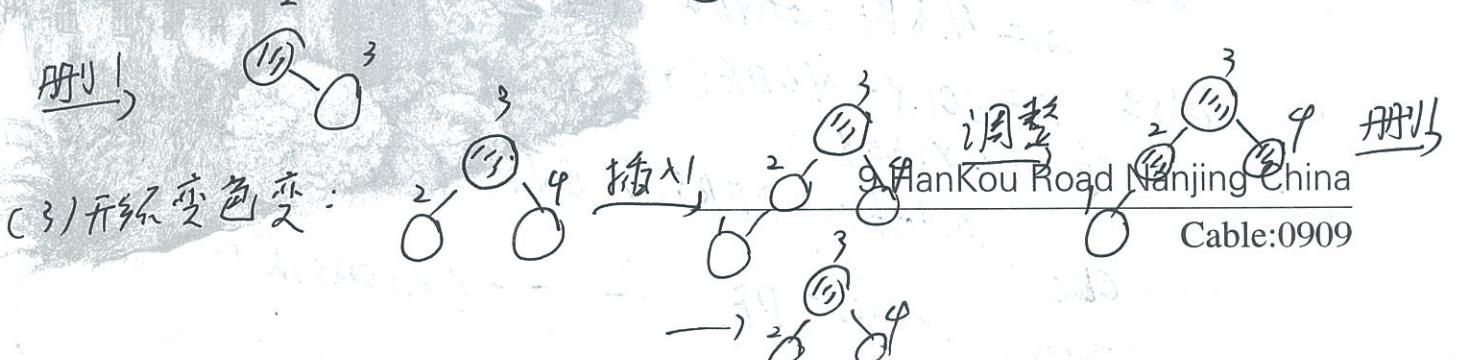
- (1) 当被删除结点无孩子时, 此时 α 为 nil, 第 2 行访问了
 (2) 当 α 的兄弟 β 无左孩子时, 在情况 1 中左旋得新的 α
 即为 nil, 在第 8 行访问

13.4-7. 解: 可能一样也可能不一样

(1) 一样:

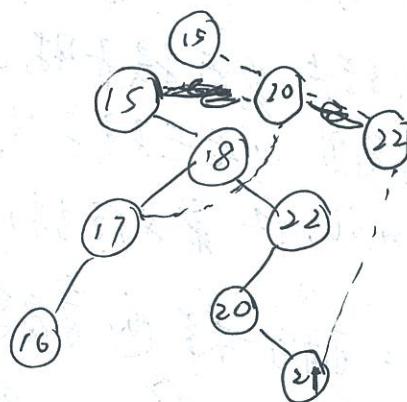
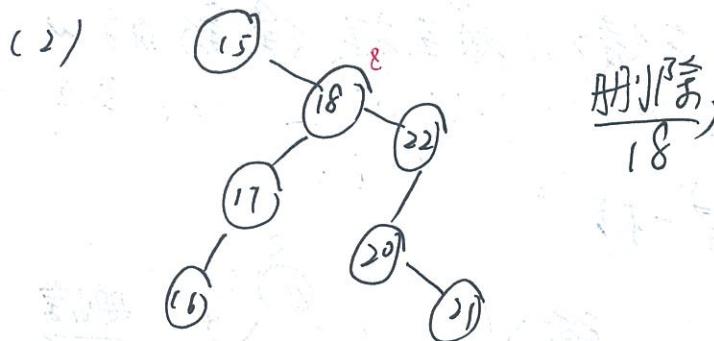
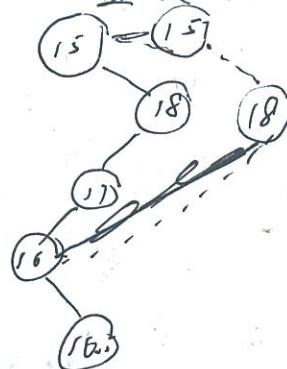
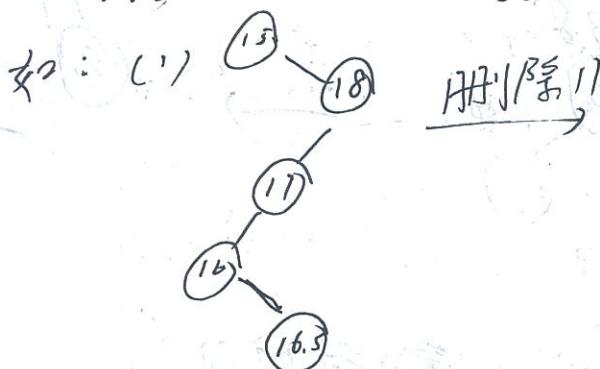


(2) 形状变化:



13-1 解：

- a. ① 当插入一个结点时，从根到该结点的路径上的所有节点都要更新
- ② 当删除一个结点时，设其后继为 y ，~~如果 y 有左子~~ 则若 y 有一个子女，则该子女取代 y ，则从根到该子女的路径上的所有节点都要更新（包括该子女结点）
若 y 有两个子女，必须从根到 y ，从 y 到 y' 的路径上的所有结点都要更新。（此时原 y 位置变为 y' ，而原 y 位置删除）



- b. 定义两个操作，MAKE-NEW-NODE(x) 创建一个键值为 x 的新节点，左子和右儿子默认为空，并返回该节点。COPY-NODE(x) 复制一个和一模一样的节点并返回， x 为根。
(包括结构)

PERSISTENT-TREE-INSERT(r, h)

if ($r == \text{NIL}$)

$x = \text{MAKE-NEW-NODE}(h)$;

else $x = \text{COPY-NODE}(r)$

if $h < r.\text{key}$

$x.\text{left} = \text{PERSISTENT-TREE-INSERT}(r.\text{left}, h)$;

else

$x.\text{right} = \text{PERSISTENT-TREE-INSERT}(r.\text{right}, h)$

return x



d. 所有结点都要复制一次，故时间复杂度和空间复杂度都为 $O(n)$

e. 由 d 知，我们不能给结点增加父结点属性。故可以在插入或删除结点时的查找过程中，把经过的结点进栈。

红黑树插入或删除调整时，最多只需取出两个、 $O(1)$ ，时间为 $O(1)$ 。
之后其它操作仿红黑树即可实现 $O(\lg n)$

13-2. 解：

a. 在插入的调整过程中，情况 1 会导致指针 α 上升，若上移后新的 α 为根结点，此时 α 为红，在 α 行被涂为黑，则树的黑高加 1。
而在删除的调整过程中，情况 2 会让 α 上升一层，若上升后 α 为根结点，由于其本应该为双重黑色，故树的黑高将减 1。
其余情况下，黑高保持不变。

其具体情况 1，黑高保持不变。
沿 α 下降时，每遇到 α 是遇到的结点为黑色，则 $bh = 1$ 。
故为 $O(1)$ 时间

一直往右子树走

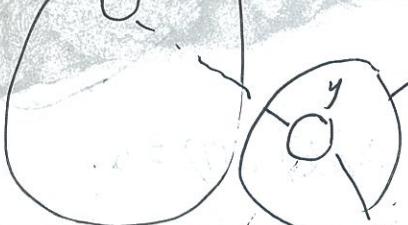
b. 从 T_1 的根开始，若遇有左子树则往左子树走，否则沿至子树走。
当遇到黑色节点时， $bh = bh - 1$ 。则当 $bh = T_2.bh$ 时，其所在结点即为所求。

c. 由于 T_1 与 T_2 黑高相同，故可令 T_1, T_2 为 α 的左右子树，并把 α 代替原来 α 的位置。注意把 α 涂为红色（由已知，此时 T_2 子树的所有结点为黑色，其它部分所有结点为白）

d. 当 α, β 为红色时，需要调整。仿红黑树的调整方法，
可在 $O(\lg n)$ 时间内调整完毕。

e. 当 $T_1.bh \leq T_2.bh$ 时，从 T_2 的根结点开始，一直往左走，直到 bh 为 $T_1.bh$ 为止。

f. 假设 $T_1.bh > T_2.bh$ ，则由 α 找到的 β ，其 β 由于 γ 位于 T_1 的左子树， β, γ 的所有结点比 β 的其余结点大。把 β 构造出的新子树，必为 β 之二叉搜索树的左子结点，故 β 的左子结点即为所求。连接后的红黑树如图。
总时间为 $O(\lg n)$



9. Hankou Road Nanjing China
 γ 为所求连接后的红黑树。
时间为 $O(\lg n)$

13-4 解：

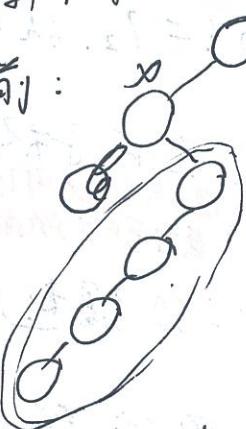
a. 问题即证：对 n 插入时按其优先级从大到小插入形成的所有二叉搜索树是唯一的，下面用归纳法证明。

方法1：假设当 $n \leq h-1$ 时树是唯一的，当 $n=h$ 时，由于根为优先级最小的元素，其左子树为小于根的元素，右子树为所有大于根 $\text{root}.\text{key}$ 的元素，则左子树、右子树元素集合互不相交，这些集合总个数 $\leq h-1$ ，故其构成的树唯一。得证当 $n=h$ 时树唯一。

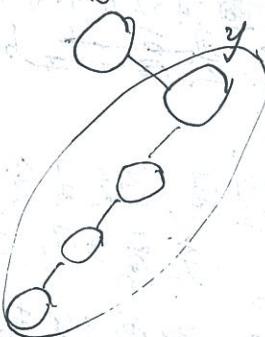
方法2：假设前 $h-1$ 个优先级插入的树唯一，当插入第 h 个元素时最大优先级的元素时，其按二叉插入左子树的方法插入，插入的位置必唯一确定，故形成的 h 个元素的树也唯一。

b. 每一棵 treap 树对应一种插入顺序，由于每个元素的优先级是随机产生的，故每一棵 treap 树与随机构造的二叉搜索树是一一对应的，故其期望高度为 $O(\lg n)$

e.e. 旋转前：

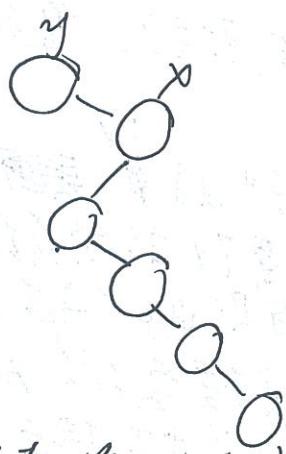


左单旋

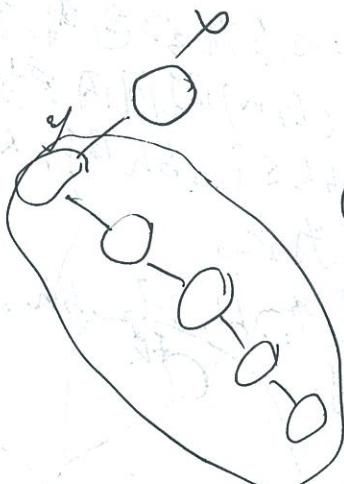


$(\text{CD})^+$

旋转前：



左单旋



$(\text{CD})^+$

综上每旋转一次， $C(D)$ 增加 1，初始 $C(D)=0$



9.



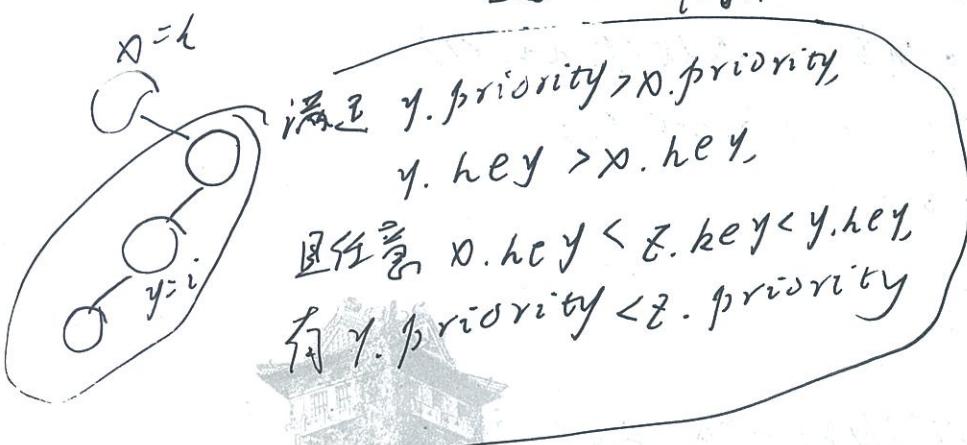
考虑 $i, i+1, \dots, h$, 由于任意 $y. key < z. key < x. key$,
有 ~~x.priority < y.priority < z.priority~~,
即任意介于 y, x 之间的元素 $z (i+1 \sim h-1)$, 其优先级必比
 x, y 高, 则对 $i \sim h$ 取立 $(h-i+1)$ 个优先级,
 x 能取最小, y 倒数第 2, 其余任取, 故概率为

$$P = \frac{(h-i+1)!}{(h-i+1)!} = \frac{1}{(h-i+1)(h-i)}$$

h. 当 $x=h$ 时, $y=i$ 取 $1 \sim h-1$, 有

$$E[C] = \sum_{i=1}^{h-1} \frac{1}{(h-i+1)(h-i)} = \sum_{j=1}^{h-1} \frac{1}{(j+1)(j)} = 1 - \frac{1}{h}$$

i. 当 $x=h$ 时, $y=i$ 取 $y+1 \sim n$, 有

$$E[D] = \sum_{i=y+1}^n \frac{1}{(i+1)(i)} = \sum_{j=1}^{n-y} \frac{1}{(j+1)j} = 1 - \frac{1}{n-y+1}$$


15.1-3. 解：修改 -7 BOTTOM-UP-CUT-ROD,

令 r 初始化为 $p[n]$, 令 i 从 1 到 $j-1$, 有

$$r[0] = 0$$

for $j=1$ to n

$$q = p[n]$$

for $i=1$ to $j-1$

$$q = \max(q, p[i] + r[j-i] - c)$$

$$r[j] = q$$

return $r[n]$

$$\left\{ \begin{array}{l} p[n] \\ p[1] + r[n-1] - c \\ p[2] + r[n-2] - c \\ \vdots \\ p[n-1] + r[1] - c \end{array} \right.$$

15.1-4. 与 EXTEND-UP-BOTTOM-UP-CUT-ROD(p, n) 不同

if $r[n] \geq 0$

return $r[n]$

if $n = 0$

$$q = 0$$

else $q = -\infty$

for $i=1$ to n

if $q < p[i] + \text{ME}$ —— aux($p, n-i, r$)

$$q = p[i] + \text{aux}(p, n-i, r);$$

$$s[n] = i;$$

$$r[n] = q$$

return q .

15.2-2. 解：修改 PRINT-OP(MATRIX-PARENTS(s, i, j)) 算法：

MATRIX-CHAIN-MULTIPLY(A, s, i, j)

if ($i = j$)

return $A[i]$;

if ($i > j$)

return $\alpha[i] * \alpha[j]$;

else

$B_1 = \text{MATRIX-}(\Delta, s, i, s[i:j])$;

$B_2 = \text{MATRIX-}(\Delta, s, s[i:j+1], j)$

return $B_1 * B_2$



15.2-4. 解: $m(1,2), m(2,3) = m(n-1, n) = m(1, 1)$
 $m(1,3), m(2,4) = m(n-2, n) \text{ 及 } m(2,2)$

$$\begin{array}{c} | \\ m(1, n-1), m(2, n) \\ m(1, n) \end{array} \quad \begin{array}{c} | \\ m(n-1, n) \end{array}$$

$$\text{若有 } n^2 + n = \frac{n(n+1)}{2} \text{ 之 } \sum$$

~~对~~于 $m(i, j)$, 具需依赖 $\begin{cases} m(i, i+1), m(i+1, j) \\ m(i, i+1), m(i+2, j) \text{ 且 } 2 \times (j-i) \text{ 条边} \\ | \\ m(i, j-1), m(j, j) \end{cases}$

~~改善数~~ 改善数 $|m(1, 2)| + |m(1, 3)| + \dots + |m(1, n)| = 2^x (1 + 2 + \dots + n-1) = (\frac{n^2}{2} - \frac{n}{2})x = n^2 \cdot x$

$|m(2, 3)| + |m(2, 4)| + \dots + |m(\cancel{2}, n)| = 2^x (1 + \dots + n-2) = (n-1)^2 - (n-1)$

|

$|m(n-1, n)| = \dots = 2^2 - 2.$

~~改善数~~ 改善数 $= (2^x - n^2) - (2^x - \cancel{1}n) = \frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2} = \frac{(n-1)n(n+1)}{3}$

其中, $m(i, j)$ 连接 $m(i, i), m(i+1, j)$
 $m(i, i+1), m(i+2, j)$

$$\begin{array}{c} | \\ m(i, j-1), m(j, j) \end{array}$$

15.2-6. 若当 $n=1$ 时成立;

假设设当 $n \leq h-1$ 时成立, 则当 $n=h$ 时

首先在任意一处加上括号, 可把算式划分为两部分,

设为 $a_1 - a_i$ 与 $a_{i+1} - a_h$, 且对于左边 $(a_1 - a_i)$ 由很

设知需 $(i-1)$ 对括号, 右边需 $(h-i-1)$ 对括号, 故 $i-1 + h-i-1 + 1 = (h-1)$ 对括号

15.3-1 解：由前段 $T(n) = \sum_{i=1}^n T(i) + cn^2$ ，
T 为 RECURSIVE - 分治法
的运算次数为 $O(n^3)$

$$T(n) \leq \begin{cases} c & n=1 \\ c + \sum_{h=1}^{n-1} (T(h) + T(n-h) + cn) = 2\sum_{i=1}^{n-1} T(i) + cn & n>1 \end{cases}$$

7. 证 $T(n) \leq cn^3$ (用代入法)

$$\begin{aligned} T(n) &\leq 2 \sum_{i=1}^{n-1} T(i) + cn \quad \text{猜想由来: } \\ &= 2(T(1) + T(2) + \dots + T(n-1)) + cn \\ &= 2(T(1) + T(2) + 2 \times 2(T(1) + T(2))) + cn \\ &= 2 \times c \times \sum_{i=1}^{n-1} i \cdot 3^{i-1} + cn = 6(T(1) + T(2)) + 2 \times 6(T(1) + T(2)) + 6(T(2) + 2(T(1) + T(2))) \\ &= c \left[\sum_{i=1}^{n-1} 2i \cdot 3^{i-1} + 2n \right] \\ &\stackrel{?}{=} (n \cdot 3^{n-1} + \frac{1}{2}(2n+1-3^n)) = \frac{2 \cdot 3^n \cdot T(1)}{3} + \frac{n+2(n-1)+2 \times 3(n-2)+\dots}{2 \times 3^2(n-3)+\dots} \end{aligned}$$

15.3-5 解：单个反例即可

长度	1	2	3	4	\vdots
价格	1	5	8	9	
限制	2	1	1	1	

$$\begin{aligned} a_1 &= 2 \\ a_2 &= a_1 + 2a_1 = 3a_1 \\ a_3 &= a_2 + 2a_2 = 3a_2 \end{aligned} \quad \begin{aligned} 1+2+2 \times 3+2 \times 3^2+\dots+2 \times 3^{n-1} \\ = 1+2 \times (1+3+\dots+3^{n-2}) \\ \approx 3^n, \text{故猜 } 3^n \times n \end{aligned}$$

无限制时最优解为切为两段长度为2, 5+5=10
有限制时最优解为1→2→3→4. 原因是两个子问题不再无关.

现在会能猜到. 因为两个子问题不再无关.

15.3-6. 解：(1) 当 $n=0$ 时， $T(0) = 1 \rightarrow n$ 分为 $1 \rightarrow h$ 与 $h \rightarrow n$ ($h=1, 2, \dots, n$)

这是一个最优子结构

(2) 当 n 为任意值时，取反例来证明：

令 $r_{12}=2, r_{13}=2.5, r_{14}=0, r_{23}=1.5, r_{24}=3, r_{34}=3$

令 $r_{12}=2, r_{13}=2.5, r_{14}=0, r_{23}=1.5, r_{24}=3, r_{34}=3$

$c_1=2, c_2=c_3=3$ 则 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ 为 $2 \times 1.5 \times 3 - 3 = 6$
而若划分子结构 $(1, 3)(3, 4)$ ，则 $1 \rightarrow 3$ 的最优解为

$1 \rightarrow 3$ ，即 $2.5 - 2 = 0.5$ ，而不是 $1 \rightarrow 2 \rightarrow 3$ ($2 \times 1.5 - 3 = 0$)，矛盾。



NANJING UNIVERSITY

PRCNU CN

15.4-2. PRINT-LCS(c, x, y, i, j)

if $i=0 \text{ or } j=0$

return

if $x[i]=y[i]$

PRINT-LCS(c, x, y, i-1, j-1)

print $x[i]$

elif $c[i-1, j] > c[i, j-1]$

PRINT-LCS(c, x, y, i-1, j)

else

PRINT-LCS(c, x, y, i, j-1)

15.4-3 解: LCS-LENGTH(X, Y)

$m = \text{length}(X)$

$n = \text{length}(Y)$

for $i=1 \text{ to } m$

for $j=1 \text{ to } n$

$c[i, j] = -1$

return LOOKUP-LENGTH(X, Y, m, n)

LOOKUP-LENGTH(X, Y, i, j)

if $c[i, j] > -1$ then

return $c[i, j]$

if $i=0 \text{ or } j=0$

$c[i, j] = 0$

else if $x[i]=y[j]$

$c[i, j] = \text{LOOKUP-LENGTH}(X, Y, i-1, j-1) + 1$

else $c[i, j] = \text{MAX}(c[i-1, j], c[i, j-1])$

~~return~~ $c[i, j]$

9. Hankou Road Nanjing China

Cable: 0909

15.4-4. (1) 假设 $n \geq 2$, 定义 $c[1, 2, 1-n]$, 其中 c 的第一行保存前一行数据, 第二行保存当前行数据。初始时, 定第一行为全 0, 开始计算第二行数据, 算完判断 $(x_i = y_j)$; 是的话令 $c[2, j] = c[1, j-1]$ 否则 $c[2, j] = \max(c[1, j], c[2, j-1])$ 。

计算完第二行后, 把该行数据复制到第一行, 重新计算第二行。

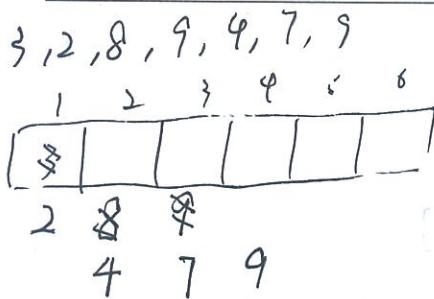
(2) 定义 $c[1, 2, 1-n]$, 计算到第 h 行每行第一次时, 定 $c[1, 2, -h-1]$ 保存前一行数据, $c[h, -n]$ 保存前一行数据, 另取一个变量 val 保存 $c[i-1, j-1]$ 数据。初始定 $c[1, 2, -h-1]$ 为全 0, 然后开始计算。若判断 $(x_i = y_j)$ 是的话令 $val = c[h]$, $c[h] = val + 1$ (此时 val 为原是的话令 $val = c[h]$, $c[h] = val$ 而 $c[h] = val + 1$, 若否, 则依然令 $val = c[h]$, 而 $c[h] = val$, 用 $temp$ 来保存), 然后 $c[h+1] = \max(c[h-1], c[h])$ 。

15.4-5. 先对这 n^2 个数据排序, 再得 y , 再找 x 和 y 的 LCS。
复杂度为 $O(n^2m) + O(n^2) = O(n^3)$

15.4-6. 先设计算法: 设序列为 $a[0, -n]$ 。
令 $c[i]$ 表示以 $a[i]$ 结尾的最长单调递增序列长度, 即 $c[i]$ 等于 $a[0 \sim i-1]$ 中所有比 $a[i]$ 小元素的最长递增子序列长度 + 1, 即 $c[i] = \max_{0 \leq h < i} (c[h] + 1)$, 若 $(a[h] \leq a[i])$ 按此算法, 依次计算 $c[0] \sim c[n]$ 。
当计算 $c[i]$ 时, 需扫描 $a[0] \sim a[i-1]$, 找每次找出比 $a[i]$ 的元素 $a[j]$ 时, 计算 $c[h+1]$ 与现有的 $c[i]$, 取大值, 复杂度为 $O(n^2)$

优化:
令 $c[i]$ 表示长度为 i 的递增子序列的最小的结束值, 则从 $0 \sim n$ 依次扫描 $a[0 \sim n]$, 遇到 $a[i]$ 时, 在数组 c 中用折半查找找出 $a[i]$ 应该插入的位置 h , 此时 $c[h] \leq a[i] < c[h+1]$ (当 $h = c.length$ 时, 说明以 $a[i]$ 结尾的某一长度递增子序列结束值比原来更小; 当 $h = c.length$ 时, 说明 $a[i]$ 比原来的 $c[c.length-1]$ 更大) (是动态递增的)

分析:



故所求为 2, 4, 7, 9.

显然，复杂度为 $\lg i + \lg 2t - t \in O(n \lg n)$

15.5.7 void printOptimalBST(i, j, r)

```
int rootChild = root[i][j];  
if (rootChild == root[i][n])  
{ cout << "h" << rootChild << endl;  
printOptimalBST(i, rootChild - 1, rootChild);  
printOptimalBST(rootChild + 1, j, rootChild);  
return; }
```

```
if (j < i - 1)  
    return;
```

```
else if (i == j - 1)  
{ if (j < r)  
    cout << "cl" << j << endl;
```

```
    else cout << "cr" << r << endl;
```

```
    return;
```

```
else if (rootChild < r)  
    cout << "h" << rootChild << endl;
```

```
    else
```

```
    printOptimalBST(i, rootChild - 1, rootChild);  
    printOptimalBST(rootChild + 1, j, rootChild);
```

9. Hankou Road Nanjing China
Cable: 0909

15.5-4. 解：

第9行修改为

if $i = j$
 $e[i, i] = e[i, i-1] + e[i+1, i] + w[i, i];$
 $\text{root}[i, i] = \infty;$

else

for $r = \text{root}[i, j-1]$ to $\text{root}[i+1, j]$

证明算法复杂度为 $O(n^2)$

对每个 $l = 1 \dots n$, 循环计算的是

$e[l, j] = e[l, l+l-1]$, 由 i 取值 $l \sim n-l+1$, 分别计算

$e[1, l], e[2, l+1], \dots, e[n-l+1, n]$

$\forall \text{root}[i, j-1] \leq \text{root}[i, j] \leq \text{root}[i+l, j]$

且 $\text{root}[i+l, j] - \text{root}[i, j-1] \geq 1$

$\text{root}[i+l, i+l-1] - \text{root}[i, i+l-2] \geq 1$

依次令 $i = l \sim n-l+1$, 得

$\text{root}(2, l) - \text{root}(1, l-1) + 1$

$\text{root}(3, l+1) - \text{root}(2, l) + 1$

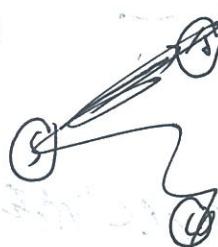
\vdots
 $\text{root}(n-l+2, n) - \text{root}(n-l+1, n-1) + 1$

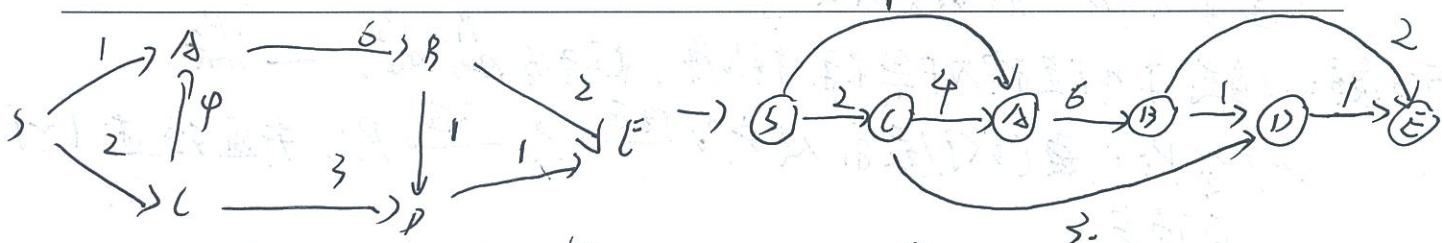
累加有 $\text{root}(n-l+2, n) - \text{root}(1, l-1) + (n-l+1)$

$\leq n-1 + n-l+1 = 2n-l < 2n$.

故 $\forall l = 1 \sim n$, 算法为 $O(n^2)$

15-1解：





用 $\text{digl}(v)$ 保存从 v 结尾的最长路径的长度，即：

$$\text{digl}(v) = \begin{cases} 0 & \text{无入边} \\ \max \text{digl}(u) + |uv|, u \rightarrow v \text{ 有边} \\ \infty & \text{无顶点} \end{cases}$$

所以我们先求出图的拓扑排序，然后按拓扑排序的顺序依次询问这些顶点，更新这些顶点连接着的边的最长路径长度，并用一个数组保存每个顶点对应的最长路径的上一个顶点，则可解之。示例：对上图，拓扑序为 $S \rightarrow C \rightarrow A \rightarrow B \rightarrow D \rightarrow E$ ，建立数组。

S	C	A	B	D	E
0	0	0	0	0	0

初始化为全0。

依次更新：	S	C	A	B	D	E
digl	0	0	0	0	0	0
path	0	0	0	0	0	0
用 \max 做 实现会更 好	2	6	12	13	14	

S	C	A	B	D	E
1	1	1	1	1	1

故最长路径为 14，对应： $S \rightarrow C \rightarrow A \rightarrow B \rightarrow E$

子问题即把所有边反向的图。

算法流程：① 求出拓扑序， $O(n + e)$

② 依次计算两个数组 $O(n + e)$

共 $O(n + e)$

5-2 解：问题可转化为：

① 把字符串 X 逆序，得 Y

② 求 X 和 Y 的最长公共子序列

15-3 解：(1) 首先对 n 个点按坐标排序，序号为 P_1, P_2, \dots, P_n

(2) 令 $P_{i,j}$ 表示 $i < j$ 表示从 P_i 向右 $\rightarrow P_j$ ，并且经过 $i \sim j$ 两点的路径。

(3) 令 $f(i, j)$ ($i \leq j \leq n$) 表示 P_i, j 的最短路径长度，且 \rightarrow 此时 $P_i \rightarrow P_j$ 的路径，最后一步为

$$f(i, j) = \begin{cases} f(i, j-1) + d_{j-1, j} & i < j-1 \\ \min_{1 \leq k < j} \{ f(k, j-1) + d_{k, j} \} & i = j-1 \end{cases}$$

为 $P_{i-1} \rightarrow P_j$

$P_i \rightarrow P_j$ 的路径，最后一步分别为 $P_{i-2} \rightarrow P_j$ -
 $P_{i-3} \rightarrow P_j$ - \dots - $P_1 \rightarrow P_j$

\rightarrow P_i 到 P_j 中，必须有一条路径经过 P_{i-1} 到 P_j 。

初值 $f(1, 2) = d_{1, 2}$

注：当 $i > j$ 时，由对称性，可知 $P(i, j) = P(j, i)$

复杂度：当 $i = j$ 时，需要计算 $f(i, i-1), f(i, i-2), \dots, f(i, 1)$ 。当 $i = j-1$ 时， $f(i, i-1), f(i, i-2), \dots, f(i, 2)$ 。

当 $i < j-1$ 时，需用到 $f(i, j-1)$

当 $i = j$ 时， $f(i, i)$ 需用到 $f(i-1, i)$

可按顺序 $\xrightarrow{\text{从左到右}} \xleftarrow{\text{从上到下}}$ 计算，再从上到下计算第二行，

$$\text{计算: } 0 + 1 + \dots + (n-2) \quad f(i-1, i)$$
$$[(n-2) + (n-3) + \dots + 1] \times f(i, j) (i < j-1)$$

$$[n-2] \quad f_{j-1} \quad f(i, i)$$

$\therefore O(n^2)$ 且 (排序需 $O(n \lg n)$)

i 1 2 3 4 5 6 7

i
1
2
3
4
5
6
7



15-9. 解：

$$\text{令} \text{extras}[i, j] = M - i + i - \sum_{h=i}^j h$$

$$l[i, j] = \begin{cases} \infty & \text{extras}[i, j] < 0 \\ 0 & \text{if } j = n \text{ 且} \text{extras}[i, j] \geq 0 \\ (\text{extras}[i, j])^3 & \text{其它} \end{cases}$$

设 $c[j]$ 表示前 j 个单词最优方案的立方和。 $\beta[j]$

$$c[j] = \begin{cases} 0 & j=0 \\ \min_{1 \leq i \leq j} (c[i-1] + l[i, j]) & (\text{第 } i \sim j \text{ 个单词放在前 } j \text{ 个单词的最后一行}) \end{cases}$$

伪代码：

PRINT-NEATLY(c, n, M)

$$\text{for } i=1 \text{ to } n \\ \text{extras}[i, i] = M - i$$

$$\text{for } j=i+1 \text{ to } n \\ \text{extras}[i, j] = \text{extras}[i, j-1] - j + 1$$

$$\text{for } i=1 \text{ to } n$$

$$\text{for } j=i \text{ to } n$$

$$\text{if } \text{extras}[i, j] < 0$$

$$l[i, j] = \infty$$

$$\text{elseif } j = n \text{ and } \text{extras}[i, j] \geq 0$$

$$l[i, j] = 0$$

$$\text{else } l[i, j] = (\text{extras}[i, j])^3$$

$$\text{else } l[i, j] = (\text{extras}[i, j])^3$$

$$c[0] = 0$$

$$\text{for } j=1 \text{ to } n$$

$$c[j] = \infty$$

$$\text{for } i=1 \text{ to } j$$

$$\text{if } c[i-1] + l[i, j] < c[j]$$

$$c[j] = c[i-1] + l[i, j]$$

$$\beta[j] = i$$

return c and β , 前 j 行得最单一行的开始字符 /

5-8. 解：从第一行开始，并

令 $B[i, j]$ 表示以像素 $A[i, j]$ 结束的接缝的最低破壞度，

$$则 B[i, j] = \begin{cases} \min(B[i-1, j-1], B[i-1, j], B[i-1, j+1]) + d[i, j] & 1 \leq j \leq n-1 \\ \min(B[i-1, 1] + B[i-1, 2]) + d[i, 1] & j=1 \\ \min(B[i-1, n] + B[i-1, n-1]) + d[i, n] & j=n \end{cases}$$

然后，我们可以从左往右，从上往下计算出所有位置的 $B[i, j]$ 。

并令 $B[n, j]$ 取出最后一行的最小的 $B[n, j]$ 即可（可用一个数组来保存每一个 $B[i, j]$ 对应的上一行中像素的纵坐标）

15-9. 解：^(参加和不参加) 表示对它的下属们所能产生的最大实际评估。

令 $f(x, 0)$ 表示 x 不参加时它的下属们所能产生的最大实际评估，则有

$$f(x, 0) = \sum_{y \in \text{son}(x)} \max\{f(y, 0), f(y, 1)\}$$

$$f(x, 1) = \sum_{y \in \text{son}(x)} \max\{f(y, 0)\} + \text{rating}(x)$$

初始：若 x 为根 $\{f, 2\}$

$$\begin{cases} f(x, 0) = 0 \\ f(x, 1) = \text{rating}(x) \end{cases}$$

复杂度 $O(V+E)$

(运行时，我们可以从根出发，依次遍历到集叶，—)

15-12. 令 $v[i, x]$ 表示在 x 美元，从 $i, i+1, \dots, n$ 位置中所能搭配自己的最大 vorp 。 $s(i, x)$ 表示第 i 位置中所有费用低于 x 美元的个数。则

$$v[i, x] = \begin{cases} \max_{\beta \in s(n, x)} \{\beta \cdot \text{vorp}\} & i=n \\ \max \left\{ v[i+1, x], \max_{\beta \in s(i, x)} \left\{ \beta \cdot \text{vorp} + v(i+1, x - \beta \cdot \text{vorp}) \right\} \right\} & i < n \end{cases}$$

↑ 等第 $i+1$ 位置的个数

令 p_i^x 表示第 i 位置补充的第 j 个



NANJING UNIVERSITY

PRCNUNI

FREE-AGENT-VORP(b, y, p_x)

let $v[1-N][0..x]$ and $who[1-N][0..x]$ be new tables
for $x=0$ to X

$$v[N, x] = -\infty$$

$$who[N, x] = 0$$

for $h=1$ to P

if $p_{Nh}.cost \leq p$ and $p_{Nh}.vorp > v[N, x]$

$$v[N, x] = p_{Nh}.vorp$$

$$who[N, x] = h$$

for $i=N-1$ down to 1

for $x=0$ to X

$$v[i, x] = v[i+1, x]$$

$$who[i, x] = 0$$

for $h=1$ to P

if $p_{ih}.cost \leq x$ and $v[i+1, x-p_{ih}.cost] + p_{ih}.vorp > v[i, x]$

$$v[i, x] = v[i+1, x-p_{ih}.cost] + p_{ih}.vorp$$

$$who[i, x] = h$$

Player Cap $v[1, x]$

JEP:

amt = x

for $i=1$ to N

$$h = who[i, amt]$$

if "h ≠ 0"

print "sign player" p_{ih}

$$amt = amt - p_{ih}.cost$$

15-11解：

$$\text{令 } D(i) = \sum_{j=1}^n d_j, D(0) = 0, D(n) = D$$

令 $f(i, j)$ 表示前 i 个字符生产 j 种机器的最低费用，且

$$f(i, j) = \min_{1 \leq k \leq i} \{ f(i-1, k) + \max_{1 \leq h \leq k} \{ 0, (D(j-h) - D(i)) \} \}$$

$$D(i) \leq j \leq D$$

复杂度 $O(nD^2)$

即依次计算 (下面的 h 应该改为 f)

$$h(1, D(1)), h(1, D(1)+1), \dots, h(1, D)$$

$$h(2, D(2)), \dots, h(2, D)$$

$$\begin{matrix} & 1 \\ & | \\ h(n, D) \end{matrix}$$

15-9解：令 $D(i)$ 表示切点之前的字符串长度，切点为

$0, 1, 2, \dots, m, m+1$
令 $f(i, j)$ 表示第 i 章 j 个切点之间的最小拆分代价。

且 $f(i, j) = \min_{0 \leq h < j} \{ f(i, h) + f(h, j) + D(j - D(i)) \}$

其中 $f(i, m+1) = 0$ 。
 $f(i, m+1)$ 表示 $f(m, m+1)$ 。

复杂度：按长度 L 从 1 至 $m+1$ 计算，有

$$\begin{matrix} L & \sim & 1 & \sim & m+1 \\ i & \sim & 0 & \sim & m+1-l \end{matrix}$$

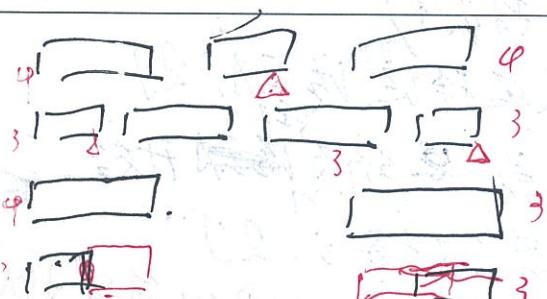
$$j = i + l$$

$$h = (i+1) \sim (j-1) \sim (L-1) / 2$$

$$\text{总复杂度为 } \sum_{l=1}^{m+1} (m+2-l) \times (L-1) = \sum_{l=0}^m (m+1-l) \times L = O(m^3)$$



16.1-3. 解：



如图，~~直接从重叠者来选择，将得列第3行~~

16.1-4. 解：(1) 首先，一种最坏解法是每次像用活动选择的算法选出最大的子集，下一次再从S-1中选出最大活动子集，依次类推。显然，最坏情况下时复杂度为 $O(n^2)$ ，而且该算法未必能选出最优解。

活动： $\{(1, 4), (2, 5), (\cancel{3, 1}), (4, 8)\}$

按上述算法将需要3间教室，实际可选 $(1, 4)(4, 8)$
再 $(2, 5), (6, 7)$

(2) 另一种算法是：① 对所有时间点（包括开始和结束）进行排序，若有其中者相同，则结束时间点在前。

证明：② 依次扫描每个时间点，遇到开始的时间点则添加一个教室到忙链表中，遇到结束时间点则取该活动所在教室从忙链表开始结点加入到忙链表中，若在忙链表中有教室，则需要结束结点减去教室加入空闲链表，若在空闲链表中有教室，则需要添加到忙链表的链表头。教室忙闲从空闲链表中取，没有并用max记录，并用max记录最大值。再创建新的。
忙闲某时刻忙闲数

证明：设用了 m 个教室，从第一个教室取出有 m 活动

同时进行，故至少需 m ，该算法有最优性。

区间图问题解法：③ 先取出一个点，再依次取与之不相交的点放入集合中，任区间已有元素相交则放进集合中

证明：假设前一个集合已达最少，则对第 $n+1$ 个



如有法！如上例

证明：假设前一个集合已达最少，则对第 $n+1$ 个

集合的元素，其无法加入任何一个集合中，故前 $n+1$ 个集合

$(1, 4)(2, 5)(6, 7)(4, 8)$

$4, 5, 6, 7, 8$

$(1, 4)(2, 5)(6, 7)$

$(1, 4)(2, 5)(6, 7)$

$(1, 4)(2, 5)(6, 7)$

9. Hankou Road Nanjing China

Cable: 0909

$(1, 4)(2, 5)(6, 7)$

$(1, 4)(2, 5)(6, 7)$

16.1-5 角色：该题是否能用动态规划求解

令 $opt[i]$ 表示 ~~前 i 个活动所能取得的最大权~~
(按结束时间对所有活动排序)

计算 $opt[i]$ 时，我们可以选择该活动， \Rightarrow

$opt[i]$ 等于 $opt[i-1]$, 其中 i 为所有 ~~所有~~ 活动开始

之前结束的活动的最后一个。若不选择 i , \Rightarrow

$opt[i] = opt[i-1]$. 例代码如下：
相似算法可以用于课本例题

Weighted-Activity-Selection(S) $O(n \lg n)$

sort S by finish time

$$opt[0] = 0$$

for $i=1$ to n :

$t = \text{binary search to find activity with}$

$\text{finish time} \leq \text{start time for } i$

$$opt[i] = \max(opt[i-1], opt[t] + w[i])$$

return $opt[n]$

16.2-2. 角色：

Dynamic-0-1-KNAPSACK(v, w, n, W)

for $w = 0$ to W

$$c[0, w] = 0$$

for $i = 1$ to n :

~~优先级队列~~ - ~~以 v 为关键字~~

$$c[i, 0] = 0$$

for $w = 1$ to W

~~用事类排序~~ - ~~以 v 为关键字~~

if $w_i \leq w$ then

if $v_i + c[i-1, w-w_i] > c[i-1, w]$

$$c[i, w] = v_i + c[i-1, w-w_i]$$

else $c[i, w] = c[i-1, w]$

else $c[i, w] = c[i-1, w]$

算法概念理解

理解



16.2-3 解：假设我们没有选择一件重量重但价值高的物品！

而是选择一件重量重但价值低的物品2，我们可以用1代替2。计算法很简单后，按重量排序后从轻到重选择即可。

16.2-4 解：

算法：每次选择 m 公里内最远的那个补水点，在那补水即已。
(也可以每到一个补水点，计算一下剩余水量是否足够到下一个点，若是，则继续前进，否则少补水)

正确性：假设存在一个最优选择，某次选择不在 m 公里内最近的补水点，则可以把它替换成前面的更小策略所选，“选择依然最优”。

16.2-7. 解：

算法：每次从集合中取出最大的 a_i 与 b_i ，即按照降序对 A, B 排序即可。

证明：任取一对 $(a_i, b_i), (a_j, b_j)$ ，若

$a_i > a_j$ 而 $b_i < b_j$ ，则可证 ~~将~~ 换成 $(a_i, b_j) (a_j, b_i)$ 后，

$$\frac{a_i^{b_i} \times a_j^{b_j}}{a_j^{b_i} \times a_i^{b_j}} = a_i^{b_j - b_i} \times a_j^{b_i - b_j} = \left(\frac{a_i}{b_j}\right)^{b_j - b_i} > 1,$$

故对任何一对 $(a_i, b_i), (a_j, b_j)$ ，若能满足 $a_i > a_j$ 且 $b_i > b_j$ ，
则可以调整得到更优解，则最后必有 ~~每~~ 第大的 a_i 与

第大的 b 相搭配。

如 A: 9, 6, 7, 8, 5

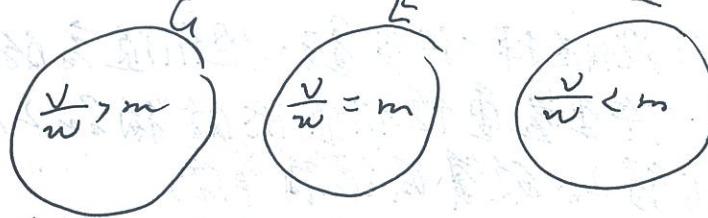
B: 4, ~~2, 5~~, 1, 1,

由于 $6 < 8$ 而 $2 > 1$ ，故交换 9, 6,
此时 A: 9, 8, 7, 6, 5 有 $8 > 7$ 而 $2 < 3$,
 $B = 4, 2, 3, 1, 1$

故交换 2, 3, 此时为最优
Cable: 0909

6.2.8. 解：

④ 求出 $\frac{v_i}{w}$ 的中位数 m , 划分为三个集合



⑤ ① 若 $W_a = W$ 继续

② $W_a > W$, 在 G 中递归

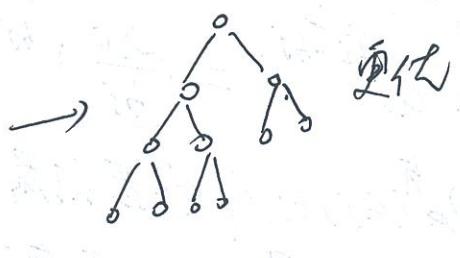
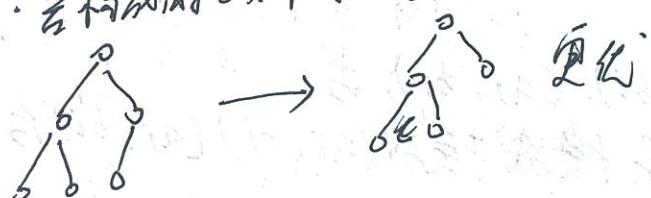
③ $W_a < W$
 ① 若 $W_a + W_E \geq W$, 则在 E 中按所给质量加入背包
 ② 若 $W_a + W_E < W$, 对 L 继续递归, 背包质量变为 $W - W_a - W_E$

复杂度：① 每次求中位数， $O(n)$

② 每次递归，规模至少减半。

$$f(n) \leq f\left(\frac{n}{2}\right) + O(n) \in O(n)$$

16.3-2. 解：若构成的二叉树如图：



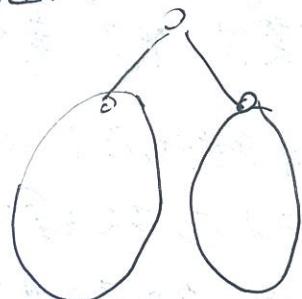
16.3-3. 解：

① 保证构建的二叉树的根结点值即为中所有叶结点频率之和。
用数学归纳法，假设该二叉树根结点

如图，
 左子树 $v_{left} = v_{x_1} + v_{x_2} + \dots + v_{x_n}$
 右子树 $v_{right} = v_{y_1} + v_{y_2} + \dots + v_{y_m}$
 则 $v = v_{left} + v_{right} = v_{x_1} + v_{x_2} + \dots + v_{x_n} + v_{y_1} + v_{y_2} + \dots + v_{y_m}$
 为所有叶结点频率之和。



② 继续证该题结论。假设左，右子树都满足结论，则



合并后，所有叶结点高度增加1，
则总的代价增加 $\Delta = v_{left} + v_{right} - v_n$
由①知 $v = v_{left} + v_{right}$

16.4-2. 试证交换性 立得证。

设 $A, B \in \Gamma$ 且 $|A| < |B|$ ，其中 $A = \{a_1, a_2, \dots, a_k\}$, $B = \{b_1, b_2, \dots, b_l\}$
若交换性成立，则 B 中任何一个元素加入 A 都会使 A 结构不变，
故 B 中任何一个元素都能由 A 线性表示，即 $\gamma(B) \leq \gamma(A)$, 且 $|B| > |A|$
故，故交换性成立。

16.4-3. 考虑图扑从阵的情形。

令 $|A'| \leq |B'|$, A' 取 $x \in (B' - A')$, 则 $A' \cup \{x\}$ 对应 $\{S - A'\} - \{x\}$,
又对任意 $y \in (B' - A')$, 必有 $y \in S - A'$ (因为 S 为全集)，
又对任意 $y \in (B' - A')$, 必有 $y \in S - A'$ (因为 S 为全集)，
问题转化为证明 $\{S - A'\}$ 中去掉某一个 x ，其依然连通。
(包含生成树) 用反证法：假定 $\{S - A'\}$ 中去掉任何 $x \in (B' - A')$
中都仍再包含生成树，则 $\{S - A'\}$ 必已为生成树，由生成树
边的数目是固定的，在 $|A'|$ 为固定，且已达最大值，与 $|B'| > |A'|$
矛盾。

16.5-2. ④ 估计数排序的方法 P_{10^9}

① 构造一个 $|A|$ 大的数据组 B , 初步化为全。

② 找出 d_i , 在 B 中填入该数字的个数 (若 $d_i > |A|$, 可省去)

③ 令 $B[i] = B[i] + B[i-1]$, 可算出 i 的 d_i 数，若

该数 i , i , ~~说明~~ 说明不独立。

16-1角至 尽可能

a. 算法：每次~~都用~~用最大值的硬币即用的

正确性：

① 为证最优算法中，最多只能有 2 个 10 美分，1 个 5 美分，4 个 1 美分且不能同时有 2 个 10 美分和 1 个 5 美分的情况。

因为 3 个 10 美分可转化为 25 分，2 个 5 美分可转化为 10 美分，1 个 1 美分可转化为 1 个 5 美分， $2 \times 10 + 1 \times 5 = 25$ ，故得证。

② 由①可知，最优算法中的 10、5、1 美分硬币，能组成的最大面值为 24。

③ 假设某种最优解中有 n' 个 25 美分，用上述贪心算法用了 n 个 25 美分，由显然 $n' \leq n$ ，若 $n' < n$ ，多出的 $n - n'$ 个 25 美分用贪心法用 10、5、1 美分的硬币组成，故 $n = n'$ 。同理，
若 n' 个 25 美分，由显然 $n' \leq n$ ，若 $n' < n$ ，多出的 $n - n'$ 个 25 美分用贪心法用 10、5、1 美分的硬币组成，故 $n = n'$ 。同理，
1 个 5 美分，4 个 1 美分最多只能组成 9 美分，故最优解中
1 个 5 美分，4 个 1 美分与贪心法相同。故依此类推，所用的 10 美分个数与贪心法相同，贪心法即为最优解。

b. 同 a 可证最多用 $(c-1)$ 个 c^h 硬币， $(c-1)c^{h-1} + \dots + (c-1)c^0$ 。

用这些硬币能构成的最大面值为 $(c-1)c^{h-1} + (c-1)c^{h-2} + \dots + (c-1)c^0$
 $= (c-1) \times \frac{1 - (1-c)^h}{1 - c} = (c-1)(1 - (1-c)^h)$ ，故同上可证贪心法即为最优解。

c. 如 $2^0, 6, 8, 1$ ，对于 8 美分，用上述算法为 $6+1+1$ ，实际最优为 $2+4$ 。

d. 用动态规划思想，

$$d(n) = \min_{1 \leq i \leq h} d(n - a_i) + 1$$

$\text{且 } a_i \leq n$



16-2 解:

a. ① 先对 p_i 排序② 按 p_i 从小到大安排启动.证明: 假设某一个最优算法排在最前的为 p_i 所有任务运行时间最长的任务, 都在第 i 位运行. 对调 p_i 与 p_j , 运行

$$\begin{array}{c} p_i \\ \hline h \quad p_i \end{array} \rightarrow \begin{array}{c} p_j \\ \hline p_i \end{array} \quad \text{时间变化为} \\ p_i \times (n-1) - (n-1)h_i - (n-1)p_i \\ = (h-1) \times (p_i - p_j) \geq 0,$$

故满足贪心选择性质

b. 每有一个任务的 r_i 到来时, 把 p_i 加入等待队列与当前运行任务的剩余时间相比, 若更短则抢占.

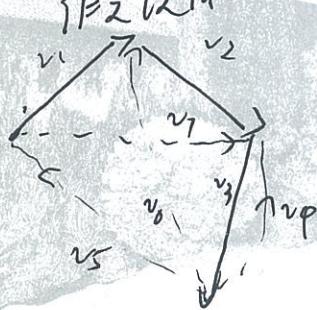
将 p_i 与 p_j 交换的 $(h-2)$ 个任务都往前了 ($p_i - p_j$), 而 p_i 与 p_j 的对调, 使任务 p_i 的完成时间之和少了 $p_i - p_j$, 故 $(h-1)k(p_i)$

16-3. 解:

a. 充分性: ~~环无奇数度顶点~~ \rightarrow 线性无关
 假设线性相关, 则必有 $(h_1 \times b_1 + h_2 \times b_2 + \dots + h_n \times b_n) \bmod 2 \equiv 0$.
 其中 h_1, h_2, \dots, h_n 全为偶数, 取 h_1, h_2, \dots, h_n 中奇数的部分, 构成的边集含每条数其等价于, 取这些 h 对应的边, 其构成的边集含的每个顶点 ~~奇数度数必为偶数~~, 即每行之和每条出去的边必有一条回来的边, 故构成环, 矛盾

必要性: 线性无关 \rightarrow 无环

假设没有环, 则这些边对应的列集合显然无关.

假设, A 为 $\{v_1, v_2, v_3\}$,B 为 $\{v_4, v_5, v_6\}$,显然 A, B 无环, 故 A, B $\in I$, 然而 $A \cup B \not\in I$ 且 B 都将形成环

9. HanKou Road Nanjing China

Cable: 0909

d. 与 a 相似



17.2-1. 解:

令 $\text{push} = 3$ 其中 push 操作中美元用于 push 本身，1 美元用在在未来 pop 的代价，另一美元用于 copy，而如 mult 中的 1 美元都用 $\text{MULT}(\text{pop}) = 3$ ，所以在 copy 上这样，经过 n 次操作后，~~累计~~ 并不积累多了几美元用于 copy 上。总为 $O(n)$ 需要吗？(金沙)

17.2-2. 解：每次操作公式多了美元的摊位还代价，

~~其中美元的代价~~对 $i=1$ 不是 2 时，付出 1 美元，存储 2 美元；对 $i=2$ 时，付出 3+之前累积的信用。我们要为 $i=2$ 时支付代价，累积的信用为 $(2^{h-1} - 1) + 1 \times 2 = 2^h - 2$ ，加上自己的美元， $2^h - 2 + 3 > 2^h$ ，不足以支付。综上，每次为 $3 = O(1)$ ，共 n 次共 $O(n)$ 。

17.2-3. 解：令 $A.\text{map}$ 指向最高位的 1，
 我们对每个 置位 操作花费 1 美元，而 reset 操作花费 1 美元。
 其中，increment 的 1 美元中 1 美元用于置位的 1，1 美元用作存储连起来
 应付未来复位的信用，1 美元用于修改 $A.\text{map}$ 的费用，另外 1 美元则
 为信用，用在 $A.\text{map}$ 增加时的 置位。这些位必须
 当进行 reset 操作时，我们需对 $1 \sim A.\text{max}$ 位置 0，此时，这些位必
 都曾经被设为 $A.\text{map}$ ，故都存储了信用，故 reset 操作共需花费

检测位的信用呢？1 美元于修改 $A.\text{map}$ 上。
 $\text{INCREMENT}(A)$

```

 $i = 0$ 
if  $i < A.\text{length}$  } 读课本
     $A[i] = 1$ 
    if  $i > A.\text{max}$ 
         $A.\text{max} = i$ 
    else  $A.\text{max} = i$ 

```

RESET(A)

```

for  $i = 0$  to  $A.\text{map}$ 
     $A[i] = 0$ 
 $A.\text{max} = -1$ 

```

考虑：1 美元：置位检测
 1 美元：未来的复位检测

0 1 1 1 1 1

↓ 检测：1 美元 \rightarrow 读场花费
 置 1：1 美元

9. Hankou Road Nanjing China

共 6 美元

↓ 检测：1 美元 Cable: 0909
 置 0：1 美元 可能不能改：1 美元
 另外 1 美元用于未来 reset 的前位

17.3-2. 参考前面 17.2-2 方法,

我们令 $c_i^* = \begin{cases} 2 & i \text{ 是 } 2 \text{ 的幂} \\ 3 & \text{其它,} \end{cases}$

则从 $2^j+1 \sim 2^{j+1}$, 其累积的 credit 为

$$(2^{j+1}-2^{j+1})+1) \times 2 + 1 = (2^{j+1}-1) \times 2 + 1 = 2^{j+1}, \text{ 加入 } 2^j \text{ 自身的 credit}$$

2^{j+1} 的 credit

故当 $i=2^j+k$ ($2^j \leq i < 2^{j+1}$) 时, 其累积的 credit 为

$$2 \times k = 2 \times (i - 2^j) = 2 \times (i - 2^{\lfloor \log_2 i \rfloor}),$$

$$\text{故 } \Delta D_i = 2 \times (i - 2^{\lfloor \log_2 i \rfloor}) \quad (i \geq 1), \text{ 且 } D_{i-1} = 0,$$

且当 $i=2^j$ 时, $c_i^* = c_i + \Delta D_i - \Delta D_{i-1}$

$$= i + 2 \times (i - i) - 2 \times (i - \frac{i}{2})$$

$$= i - i + 2 = 2 \quad (\text{注: 当 } i=2^0 \text{ 时, } c_i^* = 1 + 0 - 0 = 1)$$

当 $i=2^j+k$ 时, $c_i^* = c_i + \Delta D_i - \Delta D_{i-1}$

$$= 1 + 2 \times (i - 2^j) - 2 \times (i - 1 - 2^j)$$

$$= 1 + 2 = 3.$$

故均摊代价为 $O(1)$.

17.3-3 解: 为使得 insert 和 extract-min 的代价为 $\lg n$ 的某一个常数, 令 n_i 为第 i 次操作后堆的数目.

$$\text{令 } \Delta D_i = \begin{cases} 0 & n_i = 0 \\ kn_i \ln n_i & n_i > 0, \end{cases}$$

若证任意 $n \geq 2$, 有 $n^{\ln \frac{n}{n-1}} \leq 2$,

$$\text{由 } n^{\ln \frac{n}{n-1}} = n \ln(1 + \frac{1}{n-1}) = \ln(1 + \frac{1}{n-1})^n$$

$$\leq (ne)^{\frac{n}{n-1}} \quad (\text{由 } 1+x \leq e^x \text{ 恒成立})$$

$$= \frac{n}{n-1} \leq 2. \text{ 得证.}$$

对于 insert 操作, 有

$$c_i^* = c_i + \Delta D_i - \Delta D_{i-1}, \text{ 且 } n_i = n_{i-1} + 1$$

$$= kn_i \ln n_i + kn_{i-1} \ln n_{i-1} - kn_{i-1} \ln(n_{i-1} + 1)$$

$$= kn_i \ln n_i + kn_{i-1} \ln n_{i-1} - kn_{i-1} \ln(n_{i-1} + 1) < kn_i \ln n_i + kn_{i-1} \ln n_{i-1}$$

$$= kn_i \ln n_i$$

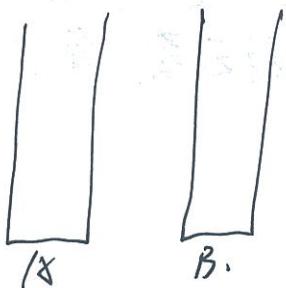
$$< 2kn_i \ln n_i$$



对于 extract-min 操作，有 $n_i = n_{i-1}$

$$\begin{aligned} \hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= k \ln n_{i-1} + k(n_{i-1}-1) (\ln(n_{i-1}-1) \cancel{\neq} \ln n_{i-1}/n_{i-1}) \\ &= k \ln n_{i-1} - k \ln(n_{i-1}-1) + \cancel{k \frac{n_{i-1}}{n_{i-1}}} \\ &\ll k \ln \frac{n_{i-1}}{n_{i-1}} < 2k = O(1) \end{aligned}$$

17.3-6. 解：



算法：插入时从 B 中插入，删除时从 A 中删除。
删除时，若 A 为空，则把所有 B 中的元素 [元素] 移到 A。

推证分析：令 $\Phi(D_i)$ = 第 i 次操作后 B 中元素个数，且 $\Phi(D_0) = 0$ ，

$$\begin{aligned} \text{插入时: } \hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + 1 = 2 = O(1) \end{aligned}$$

$$\begin{aligned} \text{删除时: } \hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= \begin{cases} 1 + 0 = 1 & \text{当 A 不为空时} \\ \Phi(D_{i-1}) + 0 - \Phi(D_{i-1}) = 0 & \text{当 A 为空时,} \end{cases} \\ &\text{故 } \hat{c}_i \leq 1 \in O(1) \end{aligned}$$

另解：核算法
每入队一个元素的代价是 3 美元，1 美元用于插入本身，1 美元用于从 B → A，第 1 美元用于删除。9. HanKou Road Nanjing China
则删除操作的代价为 0 美元。

17.3-7. 解：

算法：构建一个数组，插入时以尾部插入，删除时
先查找中位数（~~中位数~~），耗时 $O(n)$ ，再用 division 求出
所有阶位数的数，删除之。

时间复杂度分析：每（核算法）：

每个插入操作代价为 3 美元，1 美元用于插入本身，1 美元
用于找中位数和 division，最后一美元用于当元素被
删除时，把这一美元给予剩余的一半元素。

这样，删除时，每个元素还有 2 美元的信用，其中 1 美元
用于查找中位数和 division，剩下 1 美元，当删除
一半的元素时，这一美元给予剩余的元素，每个元素
信子依然还有 2 美元的信用。

17.4-3. ① 若删除后未小于 $\frac{1}{3}$ ，则

$$\begin{aligned} c_i^1 &= c_i + \lceil \log(D_i) - \log(D_{i+1}) \rceil \\ &= 1 + |2^{\lfloor \log_2(\text{num-size}) \rfloor} - 2^{\lfloor \log_2(\text{num+1-size}) \rfloor}| \\ &\leq 1 + |(2^{\lfloor \log_2(\text{num-size}) \rfloor} - 2^{\lfloor \log_2(\text{num+1-size}) \rfloor})| \quad (|a| - |b| \leq |a-b|) \\ &= 1 + 2 = 3 \end{aligned}$$

② 若删除后小于 $\frac{1}{3}$ ，则

$$\begin{aligned} c_i^1 &= (\text{num+1}) + (\text{size} - 2^{\lfloor \log_2(\text{num}) \rfloor}) - [(\text{size} - 2^{\lfloor \log_2(\text{num+1}) \rfloor})] \\ &= (\text{num+1}) + (2^{\lfloor \log_2(\text{num+1}) \rfloor} - 2^{\lfloor \log_2(\text{num}) \rfloor}) - [2^{\lfloor \log_2(\text{num+1}) \rfloor} - 2^{\lfloor \log_2(\text{num}) \rfloor}] \end{aligned}$$

17-1.b. 观察 $\text{rev}_n(\text{rev}_n(a)+1)$ 的过程，我们设 $a = abcd$ ($a, b, c, d = 0, 1$)

17.1 $\text{rev}_n(\text{rev}_n(a)+1)$

= $\text{rev}_n(dcba + 1)$

即相当于对 \overbrace{abcd}^t 从高位至低位执行 +1 的操作，
故称之为位逆序的二进制计数器。

伪代码如下：



BIT-REVERSED-INCREMEN7(A)

 $i = 0 \& A.length - 1$ while $i \geq 0$ 且 $A[i] = -1$ $A[i] = 0$ $i = i - 1$ if $i \geq 0$ $A[i] = 1$

由观察可知，题给序列为恰好为 $0, 1, 2, -1$ 的逆序。下面分析原因

设 $x = abcd$, 则 $\text{rev}_n(\text{rev}_n(x) + 1) = \underline{\underline{ab\mid cd}}$

设 $x' = \text{rev}_n(x) = dcba$, 则 $x' + 1 = \underline{\underline{dcba\mid 1}}$ 故

$\text{rev}(\text{rev}(x) + 1) = \text{rev}(x' + 1)$, 正常。

故可设初值的数为 x' ($x' = 0, 1, 2, -3$)

 $x' \xrightarrow{\text{+1}} x' + 1$

故得证。

$\begin{array}{rcl} abcd(0001) & \xrightarrow{\text{rev}(\text{rev}(x) + 1)} & abcd \\ \downarrow & \downarrow \text{rev}(x' + 1) & \downarrow \\ dcba(1000) & \xrightarrow{\text{rev}(\text{rev}(x) + 1)} & \text{rev}(abcd) \end{array}$

则在执行位逆置操作时，我们对每一位的下标执行 $-i$, $\text{rev}(\text{rev}(x) + 1)$ ，
然后把这两个下标对应的数交换，由前边知

每次二进制计算器的复杂度为 $O(1)$, 故总为 $O(n)$

每次二进制计算器的复杂度为 $O(1)$, 故总为 $O(n)$

由上知算法执行过程与位操作无关，故依此得证。

17-2. 解：

查找时，我们可以对 h^n 数组依次进行二分查找；

复杂度为： $(g_2^0 + g_2^1 + \dots + g_2^{h-1}) = 0 + 1 + \dots + (h-1) = \frac{h(h-1)}{2} \in O(g^2 n)$

b. 算法：我们可以先创建一个 l 位的数组 B 放入 A ，然后查看原数组 A 中的 ~~小~~ 小。是空的话用 B 替代 A ，结束算法，否则，我们令 A 和 B 为长度为 2 的数组，然后查看 A ，是空的话，代替 A ，不是的话继续合并（两个有序数组合并成一个有序数组的复杂度：最坏情况时，插入时前 $(k-1)$ 个数组为满，则为 $O(n)$ ）
为 $2^0 + 2^1 + 2^2 + \dots + 2^{k-1} = 2^k - 1 \in O(n)$

插值分析：

①(聚分分析) 考虑 n 个连续的插入操作，设每次插入后 ~~全局步数的第 0 步到高步数的第 0 步~~ 为第 r 位，则插入时，耗时 $2^0 + 2^1 + \dots + 2^r = O(2^{r+1})$ ，而 n 次连续插入中， $r=0$ 的情况有 $\frac{n}{2}$ 次， $r=1$ 为 $\frac{n}{2^2}$ ，依此类推， $r=n-1$ 处于第 r 位的情况有 $\frac{n}{2^n}$ 次，故其

$$\left(\sum_{r=0}^{n-1} \frac{n}{2^{r+1}}\right) \times 2^{r+1} = O(n \lg n)$$

②(核算法)：我们每插入一个数时，我们花 1 美元用于插入本身，剩余 $n-1$ 美元用于合并的过程，故平均每次为 $O(n) = O(\lg n)$

c. 算法：①找到 h^n 数组中最低位的 ~~小~~ 的数组，取出最小的那个，设为 y ，用时 $O(n)$ ，设准分子

②按 i 中方法查找出 x ，用时 $O(g^2 n)$ ，

③删除 x ，并把 y 插入 A_i 中，用时 $O(n)$

④把剩下的剩余 (2^{i-1}) 个元素按顺序分配到 A_0, A_1, \dots, A_{i-1} 中。

复杂度：用时 $O(n)$

最坏情况： $i=h-1$ 且 $j=h-2$ ，用时显然为 $O(n)$ ，下证此即为该操作后的最坏复杂度。考虑 n 连续 n 次操作，其中前 $n-1$ 次为插入，后 $\frac{n}{2}$ 次为删除（插入，删除），若前 $n-1$ 次操作后，形成的数组为全满（除了最大的那个）， $(2^0, 2^1, \dots, 2^{h-1})$ ，然后我们再插入，其变成一个数组 A_j ，用时 $2^0 + 2^1 + \dots + 2^{j-1} = 2^j - 1$ ，

再删除，其又拆分成 j 个不满的数组，用时 $O(n)$ ，

共 2^{h-1} 次操作，耗时 $O(n^2)$ ，得证。