P33

1.4 28； 1.6 18； 1.11 26
1.13 FTF TTT FTF TFF FTF
1.16 (a) n-1 非降序排序 (b) n(n-1)/2 非升序排序 (c) 0 非降序排序 (d) 3*n(n-1)/2 (交换需要做 3 次赋值) (e) O(n2) Ω (n) (f) 不可以

1.23 $$\lim_{n\to\infty}\frac{n!}{n^n}=\frac{n(n-1)(n-2)\cdots 1}{n*n*\cdots n}=0$$ ，所以不成立

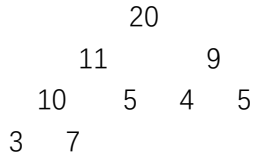
1.26 用 m=logn 替换，即 $n=2^m$. $$\lim_{n\to\infty}\frac{m!}{2^m}>0$$ ，所以 Ω (n)

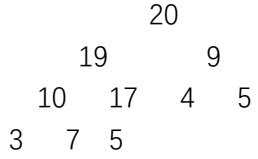
1.31 (a)第 4 行共 6 次循环，第 3 行和第 5 行合起来共（$1^2+2^2+3^2+\cdots+\log^2 n$）=1/6logn(logn+1)(2logn+1)，所以总的循环次数为 6*1/6logn(logn+1)(2logn+1)= logn(logn+1)(2logn+1); (b) Θ 更合适 (c) $\Theta(\log^3 n)$

1.35 取 3 个数，得中间元素

P86
4.3 a，b，c，d 是堆

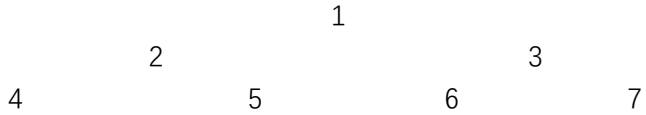
4.5 算法的思路就是一次比较根和其两个子节点，两个子节点再和其子节点比较，以此类推。复杂度是 O(n)

4.7




4.12 参考书籍：先形成树，再从最后一个非叶子节点开始调整以这个节点为根的树，依次到根节点

4.13 先画出树



8　　9　　10　　11　　12　　13　　14　　15

16　17　18　19

（a）9 次（对 1 到 9 节点为根的树进行调整）

（b）16 次，（5-9 节点调整一次，3-4 节点调整 2 次，2 节点 3 次，1 节点 1 次）

（c）1，2，3，。。。，17，18，19

4.19 一个简单的方法是将两个数组合并，然后按照形成堆的方法把数组调整成堆，O(n)。

4.23 （这道题好像我没有布置） 进行两两合并，如总共 1，2，。。。，n 个数据，先对 1 和 2，3 和 4，5 和 6，。。。，n-1 和 n 进行合并，之后对 1 和 2，3 和 4，合并后的数据进行合并，对 5 和 6，7 和 8，合并后的数据进行合并，以此类推。

4.29

1. 这两种方法进行合并以后的秩都是从 1 到 log n（对 n 个数据），也就是 O(n)。对 n 个数据依次合并的秩是 1，两两合并的秩是 log n
2. 按权重合并的期望查找时间要小，如对下面两个树进行合并：

如果是按秩合并，左边的树根指向右边树的根，合并后总的查找代价增加 7；
如果是按权重合并，右边树的根指向左边树的根，合并后总的查找代价增加 3；

但按秩合并的树的高度要小，如：

如果是按秩合并，左边的树根指向右边树的根，合并后树的高度为 2；
如果是按权重合并，右边树的根指向左边树的根，合并后树的高度为 3；

4.31

通过归纳法证明：

1. 当树的节点数 =1 时，树的 rank=0，高度为 $\log 1 = 0$，成立；当树的节点数 =2 时，树的 rank=1，高度为 $\log 2 = 1$，成立。

2. 假设两棵树的节点个数分别为 m 和 n，高度为 $rank_m$ 和 $rank_n$，满足 $rank_m \leq \log m$（$m \geq 2^{rank_m}$）和 $rank_n \leq \log n$（$n \geq 2^{rank_n}$），则按秩合并后，有以下 3 种情况：

(1) $m < n$:

1.1 $rank_m < rank_n$，则合并后树的节点个数为 $m+n$，且树的 $rank_{m+n} = rank_n \leq \log n \leq \log(m+n)$；

1.2 $rank_m \geq rank_n$，则合并后树的节点个数为 $m+n$，且树的 $rank_{m+n} = rank_m + 1 \leq \log m + 1 \leq \log 2m \leq \log(m+n)$；

(2) $m > n$:

2.1 $rank_m > rank_n$，则合并后树的节点个数为 $m+n$，且树的 $rank_{m+n} = rank_m \leq \log m \leq \log(m+n)$；

2.2 $rank_m \leq rank_n$，则合并后树的节点个数为 $m+n$，且树的 $rank_{m+n} = rank_n + 1 \leq \log n + 1 \leq \log 2n \leq \log(m+n)$；

(3) $m = n$:

则合并后树的节点个数为 $m+n$，且树的 $rank_{m+n} \leq \max(rank_n, rank_m)+1 \leq \max(\log n, \log m) + 1 = \log n + 1 = \log 2n = \log m + n$；

第 5 章

5.5

```
SearchElement(A, p, l, x)
{
If p==l
    If A[p]==x
        Return p;
Else
    m=(p+l)/2;
    SearchElement(A, p, m, x);
    SearchElement(A, m+1, l, x);
}
```

5.12

将 n 个元素分到 k 个桶中，复杂度为： O(k*n)；如果 n 个元素为整数，则可以通过求模的方式，这样复杂度就是 O(n)；

每个桶排序的复杂度为：O(n/k log n/k)，k 个桶的排序为：O(n log n/k)；

总复杂度为：O(k*n+n log n/k)

5.13

```
BuckSort(A, p,l,k) /*k 为桶的个数*/
{
If l-p<=k
    将元素分到 k 个桶中（每个桶要么没有元素，要么有一个或多个相同的元素）；/*因为 m 小于 n
    按照桶的顺序完成排序；
Else
    将元素分到 k 个桶中，第 i 个桶中元素为 A[p_i,…,l_i];
    For (i=1 to k)
        BuckSort (A, p_i, l_i);
}
```

复杂度递归式为：T(n)=kT(n/k)+kn
得：kn log n
这种递归的主要确定是需要开辟至少 n 个桶的空间

5.20

这是因为 R 的全排列是由如下形式构成：
perm(R)由$(r_1)perm(R_1)$，$(r_2)perm(R_2)$，…，$(r_n)perm(R_n)$

第一次执行递归调用实现了$(r_1)perm(R_1)$；

第二次递归调用前需要 r_2 和 r_1 进行交换，从而实现了$(r_2)perm(R_2)$；

为了使第三次递归调用实现了$(r_3)perm(R_3)$，所以需要交换回来。

5.33

设置两个指针 i 和 j，初始时，分别指向数组的两头。如果 i 和 j 所指的元素之和大于 x，则 j 指针减一操作；如果 i 和 j 所指的元素之和小于 x，则 i 指针加一操作，直到两数之和为 x 或者两指针指向同一个元素（没有找到 x）

第五章

6.5

```
Sum(A, i, j)
If i=j
   Return A[i];
Else
   Return sum(A, i,(j+i)/2)+ sum(A, (j+i)/2+1,j)
```

空间：n*logn

6.6

```
Frequency(A, i, j,x)
If i=j
   If A[i]=x
   Return 1
   Else
   Return 0;
Else
   Return Freqency(A, i,(j+i)/2,x)+ Freqency (A, (j+i)/2+1,j,x)
```

时间：T(n)=2T(n/2)+1=>T(n)=n

6.14

```
Mergesort(A, i, j)
If i=j
   Return A[i];
Else
   Mid_1=i+(j-i)/4
   Mid_2=i+(j-i)/2
   Mid_3=i+3(j-i)/4
   Mergesort(A, i, Mid_1)
   Mergesort(A, Mid_1+1, Mid_2)
   Mergesort(A, Mid_2+1, Mid_3)
   Mergesort(A, Mid_3+1, j)
   MERGE(A, i, Mid_1, Mid_2)
   MERGE(A, Mid_2+1, Mid_3,j)
   MERGE(A, i, Mid_2,j)
   Return A
```

时间：T(n)=4T(n/4)+n=>T(n)=nlog n

6.19（需要重新思考）

1…105 求中项的中项递归：因为只有 21 个中项（<44）递归一次

在 1…52 求第 17 小元素：递归一次（不能直接取中间元素）
求 1…52 中项的中项：递归一次，
在 1…25 求第 17 小元素：递归一次（排序求第 17 小元素）

总共 4 次

6.21
假设有 5 个元素：a、b、c、d、e。

首先，对（a、b、c、d）进行比较。在 a 和 b 之间进行一次比较，在 c 和 d 之间进行一次比较，将这两次比较中较大的元素再进行一次比较，则至少有 3 个元素能够确定全序关系。并且还额外的知道了另外一个元素与这 3 个元素中的一个元素的序关系。例如：a < b < c 且 d<c。

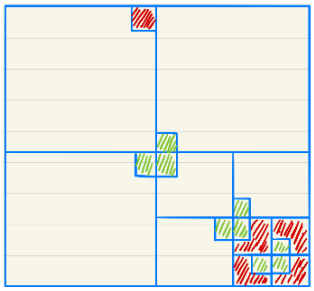
接着，我们使用折半查找将 e 插入到已序序列 a、b、c 中，至多需要 2 次比较。然后得到已序序列 a、b、c、e。

由于最开始我们已经知道了剩下的未排序元素与已序序列中一个元素的序关系，因此至少可以排除掉一个元素。相当于将剩下的未排序元素插入到一个长度为 3 的已序序列，至多需要 2 次比较。例如将 d 插入到已序序列 a、b、c、e 中，已知 d < c，则只需考虑将 d 插入到已序序列 a、b 中（这里是两个，最多 3 个）。

6.32
(a)最小或者最大时
(b)n-2，第 2 大时，第 1 大在第二个位置；第 2 小时，最小在最后一个位置

6.33
不能，在最坏的情况下是一边 1 个，另外一边 n-2 个，复杂度还是 n^2

6.54



当 k>0 时，将 $2^k \times 2^k$ 棋盘分割为 4 个 $2^{(k-1)} \times 2^{(k-1)}$ 的子棋盘。特殊方格必位于 4 个较小子棋盘之一中，其余 3 个子棋盘中无特殊方格，用一个 L 型骨牌覆盖这 3 个较小棋盘的会合处，如图所示，从而将原问题转化为 4 个较小规模的棋盘覆盖问题。递归地使用这种分割，直至棋盘简化为棋盘 1×1。

7.5

C 矩阵：

		Z	X	Y	Y	Z	X	Z
	0	0	0	0	0	0	0	0
X	0	0	1	1	1	1	1	1
Z	0	1	1	1	1	2	2	2
Y	0	1	1	2	2	2	2	2
Z	0	1	1	2	2	3	3	3
Z	0	1	1	2	2	3	3	4
Y	0	1	1	2	3	3	3	4
X	0	1	2	2	3	3	4	4

B 矩阵

		Z	X	Y	Y	Z	X	Z
X								
Z								
Y								
Z								
Z								
Y								
X								

7.7

设 A 序列为 n，B 序列为 m，

当 n<=m 时，按列压缩

```
For i=1 to m do
 for j = 1 to n do /*n 行 */
    stcur ← c[j];
    if A[i]=A[j]
          c[j] ←stpre;
    else
```

```
            c[j] ←max{c[j],c[j-1]}
    stpre ← stcur;
    end for
end for
```

当 n>m 时，按行压缩

```
For i=1 to n do
  for j = 1 to m do /*m 列 */
    stcur ← c[j];
    if A[i]=A[j]
            c[j] ←stpre;
    else
            c[j] ←max{c[j],c[j-1]}
    stpre ← stcur;
    end for
end for
```

7.9

C[1,5]=min

C[1,1]+C[2,5]+$P_0P_1P_5$=0+207+4*5*5=307

C[1,2]+C[3,5]+$P_0P_2P_5$=60+132+4*3*5=252

C[1,3]+C[4,5]+$P_0P_3P_5$=132+120+4*6*5=307

C[1,4]+C[5,5]+$P_0P_4P_5$=180+0+4*4*5=260

最优加括号为 (M1M2)((M3M4)M5)

7.16

D0= $\begin{bmatrix} 0 & 1 & \infty & 2 \\ 2 & 0 & \infty & 2 \\ \infty & 9 & 0 & 4 \\ 8 & 2 & 3 & 0 \end{bmatrix}$

D1= $\begin{bmatrix} 0 & 1 & \infty & 2 \\ 2 & 0 & \infty & 2 \\ \infty & 9 & 0 & 4 \\ 8 & 2 & 3 & 0 \end{bmatrix}$

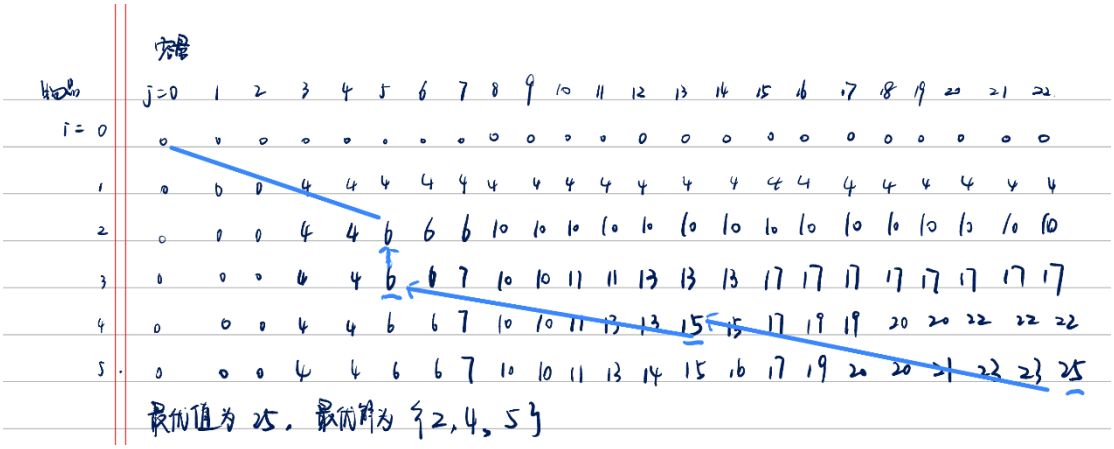
D2= $\begin{bmatrix} 0 & 1 & \infty & 2 \\ 2 & 0 & \infty & 2 \\ 11 & 9 & 0 & 4 \\ 4 & 2 & 3 & 0 \end{bmatrix}$

D3= $\begin{bmatrix} 0 & 1 & \infty & 2 \\ 2 & 0 & \infty & 2 \\ 11 & 9 & 0 & 4 \\ 4 & 2 & 3 & 0 \end{bmatrix}$

D4= $\begin{bmatrix} 0 & 1 & 5 & 2 \\ 2 & 0 & 5 & 2 \\ 8 & 6 & 0 & 4 \\ 4 & 2 & 3 & 0 \end{bmatrix}$

7.22

重量：3,5,7,8,9　价值：4,6,7,9,10



7.30

1）递归方程

$$M_y = \begin{cases} \min_i\{M_{y-v_i}+1\} & if\, y>0 \quad and \quad y-v_i>=0 \\ <0 \quad if \quad y=0 \end{cases}$$

My 表示面值为 y 时的最少硬币数目，y 面值的兑换为放入 v_i(i=1…n)和 M{y-vi}

的组合
2）时间复杂度为：ny，空间复杂度为 y

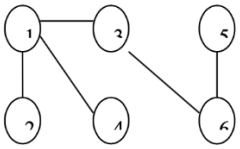
广义旅行商问题

我们通过中括号表示那些只要访问其中一个城市即可的集合，则以上例子的城市集合可写成 $\{c_1, c_2, [c_3, c_4], [c_5, c_6]\}$，用 C_i 表示可购买相同物品的集合，其中 i 可为这个集合中的任意城市，则上述例子中 C_1 表示集合 $[c_1]$，C_2 表示集合 $[c_2]$，C_3 和 C_4 都表示集合 $[c_3, c_4]$，而 C_5 和 C_6 都表示集合 $[c_5, c_6]$。参考旅行商问题的递归方程，这类旅行商问题的递归方程可写成。

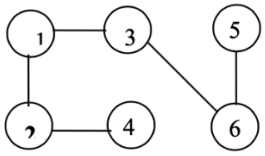
$$OPT(c_1, C, c_i) = \begin{cases} d(c_1, c_i) & |C| = 1,\ i \neq 1 \\ \min_{c_j \in C \setminus C_i} OPT(c_1, C - C_i, c_j) + d(c_j, c_i) & otherwise \end{cases} \tag{6.5}$$

8.16
设置一个变量，记录前驱。

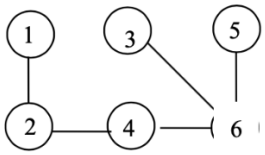
8.23
答案不唯一



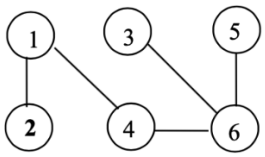
解二：



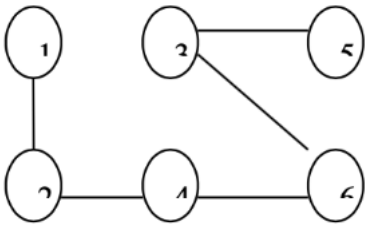
解三：



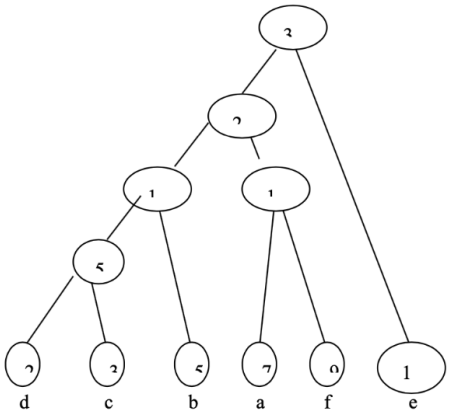
解四：



8.24



8.31



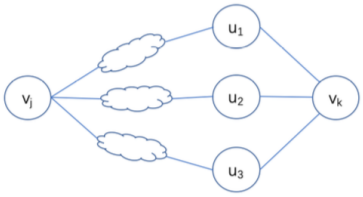
每一个二叉树都取左边为0，右边为1
则最优编码为
a:010
b:001
c:0001
d:0000
e:1
f:011
注意：编码不唯一

8.34
思路 1：
1. 对 b 以坐标 x 排序（nlogn）
2. 对 w 以坐标 x 建堆 （nlogn）
3. 依次取 b 和 w，如果能匹配则匹配，否则丢弃 b
复杂度：nlogn，并不是最优
思路 2：
1. 对每一个黑点，找出所有能支配的白点；n^2
2. 对黑点按 x 进行排序形成 B；n logn
3. 依次取 B 中的元素（设为 b），并在该元素所有能支配的白点中选取一个未匹配，且其 y 是最大的白点（设为 w），b 和 w 形成一个匹配，并标注 w 已经匹配；n^2
总复杂度 $O(n^2)$,

9.2

9.9
通过判断边的起点和终点的关系来确定，
如终点是起点的儿子，则为树边
如终点是起点的祖先，则为回边
如终点是起点的子孙，则为前向边
如没有祖先/子孙关系，则为横跨边
其中用先序号和后序号来判断：
节点 i 的先序号和后序号都小于或者大于节点 j，则节点 i 和节点 j 没有祖先/子孙关系；
否则，节点 i 的先序号小于节点 j 的先序号，则节点 i 是节点 j 的祖先（节点 i 的先序号比节点 j 的先序号小 1，则节点 i 是节点 j 的父节点；还有一种情况，如果节点 i 的先序号比节点 j 的先序号小，当后序号大 1，则节点 i 也是节点 j 的父节点）；

4. 设计一个算法计算连通图 $G=(V,E)$ 中两节点 v_i 和 v_j 间最短路径的条数。



假设从节点 v_j 到 v_k 的最短路径在就要到达 v_k 前，可沿着不同的 路径分别经由三个节点 u_1、u_2、u_3，也就是 u_1、u_2、u_3 是 v_k 在最短路径上的前 一邻节点

$$\sigma_{jk}=\sum_{u\in P_j(k)}\sigma_{ju}$$

给定一个无向图 $G=(V,E)$ 和两个顶点 u,v，让 $dist(u,v)$ 为 u 到 v 的最短路径的长度（跳数）。对于两个非空的顶点集合 $V_1,V_2\subseteq V$，且 $V_1\cap V_2=\emptyset$ 让 $dist(V_1,V_2)$ 表示两个顶点 $u\in V_1$ 和 $v\in V_2$ 之间的最短路径长度，即

$$dist(V_1,V_2)=\min\ \{dist(u,v)|u\in V_1,v\in V_2\}.$$

给出一个在时间 $O(|V|+|E|)$ 内计算出 $dist(V_1,V_2)$ 的算法思路。

解：1）在原图 G 的基础上，任意添加两个点 s 和 t，让 s 连接 V1 中所有的点，t 连接 V2 中所有的点，从而构造图 G', G'=(V',E')

2）在 G'上运行 BFS（广度优先搜索），得出 s 到 t 的最短路径 δ_{st}，则 $dist(V_1, V_2) = \delta_{st} - 2$