Hw 1

Page12：2.1-2, 2.1-4

Page16：2.2-2, 2.2-3

2.1

2.1-2 重写过程 INSERTION-SORT，使之按非升序(而不是非降序)排序。

书上非降序排序：

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1..j-1].
4      i = j - 1
5      while i > 0 and A[i] > key
6          A[i+1] = A[i]
7          i = i - 1
8      A[i+1] = key
```

非升序排序：只要对上面第5行修改：A[i] < key

2.1-4 考虑把两个 n 位二进制整数加起来的问题，这两个整数分别存储在两个 n 元数组 A 和 B 中。这两个整数的和应按二进制形式存储在一个$(n+1)$元数组 C 中。请给出该问题的形式化描述，并写出伪代码。

输入：存储两个n位二进制数的n元数组A[1..n], B[1..n]

输出：存储A, B对应二进制数的和的n+1元数组C[1..n+1]

伪码：

```
BinaryAdd(A, B):
# 输入：A[1..n], B[1..n]。从数组最高位A[n], B[n]开始存二进制数。
# 输出：C[1..n+1]。也是从最高位C[n+1]开始存输入两个二进制数的和。
    # carry表进位
    carry = 0
    # 二进制从数组的最高位开始存，所以i=n to 1，循环体内用C[i+1]
    for(i=n to 1):
        sum = A[i]+B[i]+carry
        #  应该是C[i+1]，不是C[i]
        C[i+1] = sum % 2
        carry = floor(sum/2)
    # 最后要将C[1]置为carry
    C[1] = carry
```

- 如果用C[i]存carry，最后就不用C[1]=carry
- 也可以将二进制数倒着存，即从数组最低位开始存，其实这样更简单

法2：如果if-else分类讨论，则要考虑sum=0, 1, 2, 3的情况。

2.2

2.2-2 **考虑排序存储在数组 A 中的 n 个数：首先找出 A 中的最小元素并将其与 $A[1]$中的元素进行交换。接着，找出 A 中的次最小元素并将其与 $A[2]$中的元素进行交换。对 A 中前 $n-1$ 个元素按该方式继续。该算法称为选择算法，写出其伪代码。该算法维持的循环不变式是什么？为什么它只需要对前 $n-1$ 个元素，而不是对所有 n 个元素运行？用 Θ 记号给出选择排序的最好情况与最坏情况运行时间。**

升序，选择排序

1. 伪码：

   ```
   SelectionSort(A):
   # 输入：A[1..n]
   # 输出：升序排好的A[1..n]
       n = len(A)
       # 最后一个元素不用排，所以是i to n-1
       for i = 1 to n-1:
           minId = i
           # 找i之后的元素更新minId，所以是从i+1开始
           for j = i+1 to n:
               if A[j] < A[minId]:
                   minId = j
           swap(A, i, minId)
   ```

2. 循环不变式（证明算法正确性）：
   1. 初始化
   2. 保持：排序好的部分保持有序
   3. 终止：能正确中止
3. 因为最后一个元素一定是最大的
4. 最好最坏都要依次在n-1, n-2,..,1个元素中找最小元素，然后交换，时间$\Sigma_{i=1}^{n-1} i = n(n-1)/2$

   $\theta(n^2)$

2.2-3 再次考虑线性查找问题(参见练习 2.1-3)。假定要查找的元素等可能地为数组中的任意元素，平均需要检查输入序列的多少元素？最坏情况又如何呢？用 Θ 记号给出线性查找的平均情况和最坏情况运行时间。证明你的答案。

2.1-3 考虑以下**查找问题**:

输入：n 个数的一个序列 $A=\langle a_1, a_2, \cdots, a_n\rangle$ 和一个值 v。

输出：下标 i 使得 $v=A[i]$ 或者当 v 不在 A 中出现时，v 为特殊值 NIL。

写出**线性查找**的伪代码，它扫描整个序列来查找 v。使用一个循环不变式来证明你的算法是正确的。确保你的循环不变式满足三条必要的性质。

1. 平均需要检查：

   v出现的位置求个期望：n个元素每个出现的概率都是$1/n$，故期望位置$\Sigma_{i=1}^{n} i/n = (n+1)/2$

   最坏需要检查n个元素

2. 平均和最坏都是 $\theta(n)$。证明：由 *1* ，平均和最坏都要检查$\theta(n)$量级的元素，一次检查耗时$\theta(1)$.

9月24日随堂测试

已知定理：

f(x) 是任何连续单调上升函数，且 f(x) 在整数点才可能取整数值，则：

1） $$\lfloor f(x) \rfloor = \lfloor f(\lfloor x \rfloor) \rfloor$$

2） $$\lceil f(x) \rceil = \lceil f(\lceil x \rceil) \rceil$$

证明：

$$\left\lceil \frac{\lceil n/a \rceil}{b} \right\rceil = \left\lceil \frac{n}{ab} \right\rceil \qquad \left\lfloor \frac{\lfloor n/a \rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor$$

（二者证明一个即可）

取$f(x) = x/b$，$x = n/a$即可。

HW2

2.3

Page22：2.3-3，2.3-5

2.3-3 使用数学归纳法证明：当 n 刚好是 2 的幂时，以下递归式的解是 $T(n)=n\lg n$。

$$T(n) = \begin{cases} 2 & 若\ n=2 \\ 2T(n/2)+n & 若\ n=2^k, k>1 \end{cases}$$

（1）$n = 2$时，$T(2) = 2 = 2\lg 2 = 2$

$k = 2$时，$n = 2^2 = 4$，$T(4) = 2T(2) + 4 = 8 = 4\lg 4$

（2）设$n=2^k$时，$T(2^k)=2T(2^{k-1})+2^k=2^k\lg 2^k=k2^k \qquad (*)$

证 $n=2^{k+1}$时，$T(2^{k+1})=2T(2^k)+2^{k+1}=2^{k+1}\lg 2^{k+1}=(k+1)2^{k+1}$，

由(*)式，$T(2^{k+1})=2k2^k+2^{k+1}=(k+1)2^{k+1}$.

由（1）（2）得证。

2.3-5 **回顾查找问题(参见练习 2.1-3)，注意到，如果序列 A 已排好序，就可以将该序列的中点与 v 进行比较。根据比较的结果，原序列中有一半就可以不用再做进一步的考虑了。二分查找算法重复这个过程，每次都将序列剩余部分的规模减半。为二分查找写出迭代或递归的伪代码。证明：二分查找的最坏情况运行时间为 $\Theta(\lg n)$。**

2.1-3 **考虑以下查找问题：**

输入：n 个数的一个序列 $A=\langle a_1, a_2, \cdots, a_n\rangle$和一个值 v。

输出：下标 i 使得 $v=A[i]$或者当 v 不在 A 中出现时，v 为特殊值 NIL。

写出线性查找的伪代码，它扫描整个序列来查找 v。使用一个循环不变式来证明你的算法是正确的。确保你的循环不变式满足三条必要的性质。

二分查找递归伪码：

```
BinarySearch(A, v, low, high):
    # 二分查找递归算法
    # 输入：数组A[1..n]，待查找的值v，查找区间[low,high]
    # 输出：若找到，则返回一个匹配元素的下标i，若v不在A中，则返回NIL

    mid = floor((low+high)/2)
    # 注意是low≤high，v=A[high]时，low==high
    if low <= high:
        if v == A[mid]:
            return mid
        if v > A[mid]:
            BinarySearch(A, v, mid+1, high)
        else:
            BinarySearch(A, v, low, mid-1)
    return NIL
```

题目要求证明，至少要写出递归式。

最坏情况：v不在A中，或最后一次比较才命中

时间$T(n)=T(n/2)+\theta(1)$，解为$T(n)=\theta(\lg n)$，得证。

3.1

Page31：3.1-2，3.1-4，3.1-6

3.1-2 **证明：对任意实常量 a 和 b，其中 $b>0$，有**

$$(n+a)^b=\Theta(n^b) \tag{3.2}$$

法1.

记$f(n)=(n+a)^b$，$g(n)=n^b$，即证 $n\geq n_0$时，$0\geq c_1 g\leq f\leq c_2 g$。

> b>0，保证x^b单增。注意a为负的情况。

Note that

$$\begin{aligned} n + a &\le n + |a| \\ &\le 2n \qquad \text{when } |a| \le n \,, \end{aligned}$$

and

$$\begin{aligned} n + a &\ge n - |a| \\ &\ge \frac{1}{2}n \qquad \text{when } |a| \le \tfrac{1}{2}n \,. \end{aligned}$$

Thus, when $n \ge 2|a|$,

$$0 \le \frac{1}{2}n \le n + a \le 2n \,.$$

Since $b > 0$, the inequality still holds when all parts are raised to the power b:

$$0 \le \left(\frac{1}{2}n\right)^b \le (n+a)^b \le (2n)^b \,,$$

$$0 \le \left(\frac{1}{2}\right)^b n^b \le (n+a)^b \le 2^b n^b \,.$$

Thus, $c_1 = (1/2)^b$, $c_2 = 2^b$, and $n_0 = 2|a|$ satisfy the definition.

法2. 一个具有启发意义的做法，by No.37

极限方法

由于

$$\lim_{n\to+\infty}\frac{(n+a)^b}{n^b}=\lim_{n\to\infty}\left(1+\frac{a}{n}\right)^b=1$$

即

$$\forall\varepsilon,\exists N,\forall n>N,\ 1-\varepsilon\leqslant\frac{(n+a)^b}{n^b}\leqslant 1+\varepsilon$$

取 $\varepsilon=0.05$，故取 $c_1=0.95, c_2=1.05$ 时，有

$$\exists N,\forall n>N,\ 0.95\leqslant\frac{(n+a)^b}{n^b}\leqslant 1.05$$

即为

$$\exists n_0,\forall n>n_0,\ 0.95\cdot n^b\leqslant(n+a)^b\leqslant 1.05\cdot n^b$$

故 $(n+a)^b=\Theta\left(n^b\right)$

3.1-4 $2^{n+1}=O(2^n)$成立吗？$2^{2n}=O(2^n)$成立吗？

（1）证$2^{n+1}=O(2^n)$

即证$n\geq n_1$时，$2^{n+1}\leq c_1 2^n$，不妨取$n_1=1,c_1=2$。

（2）证$2^{2n}\neq O(2^n)$

即证不存在n_2,c_2，使得$n\geq n_2$时，$2^{2n}\leq c_2 2^n$。 因为要满足$c_2\geq 2^n\to\infty$，故c_2不存在，得证。

3.1-6 **证明：一个算法的运行时间为 $\Theta(g(n))$ 当且仅当其最坏情况运行时间为 $O(g(n))$，且其最好情况运行时间为 $\Omega(g(n))$。**

$T=\theta(g)\Leftrightarrow n\geq n_0$时，$c_2 g\leq T\leq c_1 g$.　(命题1)

最坏时间$O(g)\Leftrightarrow n\geq n_1$时，$T\leq c_1' g$　(命题2.1)

最好时间$\Omega(g)\Leftrightarrow n\geq n_2$时，$T\geq c_2' g$　(命题2.2)

让$n_0=\max\{n_1,n_2\},c_1=c_1',c_2=c_2'$，则命题1 $\Leftrightarrow$ 命题2.1+2.2。

3.2

Page34：3.2-1，3.2-4，3.2-5

3.2-1　证明：若 $f(n)$和 $g(n)$是单调递增的函数，则函数 $f(n)+g(n)$和 $f(g(n))$也是单调递增的，此外，若 $f(n)$和 $g(n)$是非负的，则 $f(n)\cdot g(n)$是单调递增的。

设$x\le y$，则$f(x)\le f(y), g(x)\le g(y)$，$f(x)+g(x)\le f(y)+g(y)$，$f(g(x))\le f(g(y))$，即$f+g$和$f(g)$单增。

$f,g\ge 0$时，$0\le f(x)g(x)\le f(y)g(y)$，故$f,g$非负时，$f\cdot g$单增。

***3.2-4　函数$\lceil\lg n\rceil!$ 多项式有界吗？函数$\lceil\lg\lg n\rceil!$ 多项式有界吗？**

> 这题比较复杂。
>
> 首先，多项式有界的定义：f多项式有界（上界）$\Leftrightarrow \exists\, n_0, c, k$，使$n\ge n_0$时，$f(n)\le cn^k$。

法1

考虑到直接证明阶乘有界很困难，

引理　f 多项式有界 $\Leftrightarrow$ $\lg f(n) = O(\lg n)$

记$r=\lceil\lg n\rceil!$，$s=\lceil\lg\lg n\rceil!$，根据 *引理* ，即判断 $\lg r$和$\lg s$是否等于$O(\lg n)$。

因为：

- $\lg n! = \theta(n\lg n)$：这是因为$n! = \theta(n^n)$，同时取lg。
- $\lceil\lg n\rceil = \theta(\lg n)$

（1）

$$\begin{aligned}\lg(\lceil\lg n\rceil!) &= \Theta(\lceil\lg n\rceil\lg\lceil\lg n\rceil)\\ &= \Theta(\lg n\lg\lg n)\\ &= \omega(\lg n)\,.\end{aligned}$$

所以，$\lg r\ne O(\lg n)$，r不是多项式有界。

（2）

$$\begin{aligned}\lg(\lceil\lg\lg n\rceil!) &= \Theta(\lceil\lg\lg n\rceil\lg\lceil\lg\lg n\rceil)\\ &= \Theta(\lg\lg n\lg\lg\lg n)\\ &= o((\lg\lg n)^2)\\ &= o(\lg^2(\lg n))\\ &= o(\lg n)\,. \qquad (*)\end{aligned}$$

(*) 式是因为对$a,b>0$，$\lg^b = o(n^a)$。

所以，$\lg s = O(\lg n)$，s多项式有界。

引理 的证明：

（1）证 f 多项式有界 $\Rightarrow$ $\lg f(n) = O(\lg n)$

f多项式有界，则$\exists\, n_0, c, k$，$n\ge n_0$时，$f(n)\le cn^k$。所以，$\lg f(n)\le kc\lg n$，即$\lg f(n) = O(\lg n)$。

（2）证 f 多项式有界 $\Leftarrow$ $\lg f(n) = O(\lg n)$

（1）的证明倒过来即证（2）。

法2

有的同学想用Stirling公式$n! = \sqrt{2\pi n}(\frac{n}{e})^n(1+\theta(\frac{1}{n}))$直接证有界无界，请参考以下三位同学的证明：

No. 58：r用的Stirling，s没用，但非常漂亮地给出了s的多项式界。

3.2-4 $\lceil \lg n \rceil ! = \sqrt{2\pi m}\,(\frac{m}{e})^m e^{\alpha m}$ (反证) $\le C\cdot n^k \le C \cdot 2^{mk}$. $\forall m \ge m_0$ (常数k).

$(\frac{m}{e})^m \le C\cdot 2^{mk}$. 矛盾! ∴$\lceil \lg n \rceil !$ 多项式无界.

$\lceil \lg\lg n \rceil = m$. $m! < m^m < (2^m)^m < 2^{2^{m-1}}$

又 $\lg\lg n \ge m-1 \Rightarrow n \ge 2^{2^{m-1}} > m!$ ∴$\lceil \lg\lg n \rceil !$ 多项式有界.

其实，r 非多项式有界直接给个反例就行（by No.43）：设$n=2^k$，$r(n)=k!$，非多项式有界。

No. 47：r,s均用Stirling证的。

3.2-4 i. $n! > \sqrt{2\pi n}(n/e)^n$，$\sqrt{2\pi n}(n/e)^n \uparrow$，$n! < 2\sqrt{2\pi n}(n/e)^n$

$\lceil \lg n \rceil \ge \lg n$，

$\lceil \lg n \rceil ! > \sqrt{2\pi \lceil \lg n \rceil}\,(\frac{\lceil \lg n \rceil}{e})^{\lceil \lg n \rceil} \ge \sqrt{2\pi \lg n}\,(\frac{\lg n}{e})^{\lg n}$

$\lg(\lceil \lg n \rceil !) >$ $\lg(\sqrt{2\pi \lg n}\,(\lg n/e)^{\lg n}) = \text{Const} + \frac{1}{2}\lg^{(2)} n + \lg n \lg \frac{\lg n}{e}$，

记$f(n) :=$ $\lceil \lg n \rceil !$ 若 $f(n) = O(n^a), a>0$ 多项式有界，$\exists c, n_0$，$\forall n \ge n_0$，$0 \le f(n) \le cg(n)$，$g(n) = n^a$，

有 $\lg f(n) \le a\lg n + c$，

而 $\lg \frac{\lg n}{e}$ 无上界，对确定的 a，$(\lg \frac{\lg n}{e} - a)\lg n$ 无上界，

即对 $\forall c$，足够大的n使 $\lg f(n) > a\lg n + c$，

$\lceil \lg n \rceil !$ 非多项式有界.

ii. $\lceil \lg\lg n \rceil \le \lg(2\lg n)$，

$\lceil \lg\lg n \rceil !$ ~~$\lceil \lg n \rceil$~~$! < 2\sqrt{2\pi \lceil \lg\lg n \rceil}\,(\frac{\lceil \lg\lg n \rceil}{e})^{\lceil \lg\lg n \rceil} \le 2\sqrt{2\pi \lg(2\lg n)}\,(\frac{\lg(2\lg n)}{e})^{\lg(2\lg n)}$ (*)

$\lg(2\sqrt{2\pi \lg(2\lg n)}\,(\frac{\lg(2\lg n)}{e})^{\lg(2\lg n)}) = \text{Const} + \frac{1}{2}\lg^{(2)}(2\lg n) + \lg(2\lg n)\lg\frac{\lg(2\lg n)}{e}$，

$\frac{1}{2}\lg$ $\overset{\lg x < x}{<}$ $\text{Const} + \frac{1}{2}(1+\lg\lg n)$ ~~$\lg n$~~ $+ (1+\lg\lg n)^2$ （在 $1+\lg\lg n > 0$ 时）

要证明 $(\lg\lg n)^2 = o(\lg n)$，故 $\forall a>0$，$\exists n_0$，使 $\forall n > n_0$，

$\lg(\lceil \lg\lg n \rceil !) <$ $\text{Const} + \frac{1}{2}(1+\lg\lg n)$ ~~$\lg n$~~ $+ (1+\lg\lg n)^2 <$ ~~$a\lg n + \text{Const}$~~ $a\lg n + c$，c也是任意的，

$\lceil \lg\lg n \rceil ! = o(n^a)$，$\forall a>0$，则$\lceil \lg\lg n \rceil !$具有多项式上界，

***3. 2-5　如下两个函数中，哪一个渐近更大些：$\lg(\lg^* n)$还是 $\lg^*(\lg n)$？**

证明：设$2^{2^{\cdot^{\cdot^{2}}}}$共$k$个2，记作$\overset{k}{2}$。记$r=\lg(\lg^* n)$，$s=\lg^*(\lg n)$。

因为$\overset{k-1}{2} < n \le \overset{k}{2}$时，$\lg^* n = k$，故只要判断$n=\overset{k}{2}$，$r,s$哪个渐进更大即可。

因为$r(n)=\lg k$，$s(n)=\lg^* \overset{k-1}{2} = k-1$，明显$s$渐进更大。严格证明：

$\lim\limits_{k\to\infty}\frac{r(n)}{s(n)} = \lim\limits_{k\to\infty}\frac{\lg k}{k-1} = 0$，故$s$渐进更大。

HW3

4.1

Page42：4.1-2，4.1-5

4.1-2　对最大子数组问题，编写暴力求解方法的伪代码，其运行时间应该为 $\Theta(n^2)$。

对每个元素A[i]，求出以i开始，以j结束的所有子数组和（j=i..n-1，A[0..n-1]）。

运行时间，元素A[i]要求n-1-i+1=n-i个和，故$T(n) = \Sigma_{i=0}^{n-1} n-i = \Sigma_{i=1}^{n} i = \theta(n^2)$.

```
# ============= Brute Force =============
# O(n^2)
def MaxSubarrary_BruteForce(A):
    # 输出：最大子数组和maxSum，相应左右下标maxI,maxJ
    n = len(A)
    maxSum = A[0]
    maxI, maxJ = 0,0
    for i in range(n):
        curSum = A[i]
        if curSum > maxSum:
            maxSum = curSum
            maxI, maxJ = i, j
        for j in range(i+1,n):
            curSum += A[j]
            if curSum > maxSum:
                maxSum = curSum
                maxI, maxJ = i, j
    return maxSum, maxI, maxJ
```

4.1-5　使用如下思想为最大子数组问题设计一个非递归的、线性时间的算法。从数组的左边界开始，由左至右处理，记录到目前为止已经处理过的最大子数组。若已知 $A[1..j]$的最大子数组，基于如下性质将解扩展为 $A[1..j+1]$的最大子数组：$A[1..j+1]$的最大子数组要么是 $A[1..j]$的最大子数组，要么是某个子数组 $A[i..j+1]$($1 \leqslant i \leqslant j+1$)。在已知 $A[1..j]$的最大子数组的情况下，可以在线性时间内找出形如 $A[i..j+1]$的最大子数组。

分析如下代码写出算法思想和伪码：

```
# =========== O(n)方法的两个版本 ===========


def FindMaxSubArray(A, n):
    # 完全按照课上思路走的O(n)方法

    # max变量记录全局最大sum和下标
    # maxSum初始为A[0]而不是0，应对全为负数的数组
    maxSum = A[0]
    maxI, maxJ = 0, 0
    # current变量记录当前最大sum和下标
    curSum = 0
    curI, curJ = 0, 0
    for k in range(n):
```

```python
        curSum += A[k]
        curJ = k
        if curSum > maxSum:
            maxSum = curSum
            maxI, maxJ = curI, curJ
        # curSum如果 ≤ 0，对后面的贡献非正，舍弃，从k+1重新开始
        if curSum <= 0:
            curI = curJ = k+1
            # 赋值为0，相当于舍弃之前的和
            curSum = 0
    print(f"max = {maxSum} in [{maxI},{maxJ}]")


def FindMaxSubArray2(A, n):
    # O(n)方法的另一个版本，与上个版本思想是一致的
    # 这个版本更直接反映题意

    # max变量记录全局最大sum和下标
    # # maxSum初始为A[0]而不是0，应对全为负数的数组
    maxSum = A[0]
    maxI, maxJ = 0, 0
    # maxSumEndAtRightBound变量记录迭代过程中，以右边界结尾的最大子数组和
    maxSumEndAtRightBound = 0
    mseI, mseJ = 0, 0
    for k in range(n):
        # 由上次迭代的maxSumEndAtRightBound，确定这次迭代的maxSumEndAtRightBound
        if maxSumEndAtRightBound + A[k] > A[k]:
            maxSumEndAtRightBound += A[k]
            mseJ = k
        else:
            maxSumEndAtRightBound = A[k]
            mseI = mseJ = k
        # 由上次迭代的maxSum，即A[ ..k-1]的最大子数组和
        # 以及这次迭代的maxSumEndAtRightBound
        # 确定这次迭代的maxSum
        if maxSumEndAtRightBound > maxSum:
            maxSum = maxSumEndAtRightBound
            maxI, maxJ = mseI, mseJ
    print(f"max = {maxSum} in [{maxI},{maxJ}]")
```

非递归显然，就扫一遍，$T(n) = O(n)$，线性时间。

为什么符合题目所给的思想：第k次迭代开始，maxSum存的还是上次迭代的最大子数组和，计算以k结尾的最大子数组和maxSumEndAtRightBound，比较得第k次迭代的最大子数组和更新maxSum。

伪码略

4.3

Page50：4.3-4，4.3-6

4.3-4　证明：通过做出不同的归纳假设，我们不必调整归纳证明中的边界条件，即可克服递归式(4.19)中边界条件 $T(1)=1$ 带来的困难。

原证明：当无$T(1) = 1$限制时，

我们可以用代入法为递归式建立上界或下界。例如，我们确定下面递归式的上界：

$$T(n)=2T(\lfloor n/2\rfloor)+n \tag{4.19}$$

该递归式与递归式(4.3)和(4.4)相似。我们猜测其解为 $T(n)=O(n\lg n)$。代入法要求证明，恰当选择常数 $c>0$，可有 $T(n)\leqslant cn\lg n$。首先假定此上界对所有正数 $m<n$ 都成立，特别是对于 $m=\lfloor n/2\rfloor$，有 $T(\lfloor n/2\rfloor)\leqslant c\lfloor n/2\rfloor\lg(\lfloor n/2\rfloor)$。将其代入递归式，得到

$$\begin{aligned}T(n)&\leqslant 2(c\lfloor n/2\rfloor\lg(\lfloor n/2\rfloor))+n\leqslant cn\lg(n/2)+n\\&=cn\lg n-cn\lg 2+n\\&=cn\lg n-cn+n\leqslant cn\lg n\end{aligned}$$

其中，只要 $c\geqslant 1$，最后一步都会成立。

增加$T(1)=1$后，$n=1$时，$n\log n=0$，不存在c使$T(1)\leq n\log n=0$。

故要把假设的界放大，重新假设$T(n)=O(n\lg n+a)$，设$T(n)\leq c(n\lg n+a)$在$n/2$时成立，目标证$T(n)\leq c(n\lg n+a)$

由递推式，$T(n)=2T(\frac{n}{2})+n\leq 2c(\frac{n}{2}\log\frac{n}{2}+a)+n=cn\log n-cn+2ca+n$，这要证明这个界比目标界紧即可，即$-cn+2ca+n\leq ca$，

$c=1$时，$2c\leq ca$不可能成立。$c\geq 2$时，左边$\leq 2ca-n$，渐进小于右边。故$c\geq 2$时，$T(n)\leq c(n\lg n+a)$，$T(n)=O(n\lg n+a)=O(n\lg n)$。

4.3-6 证明：$T(n)=2T(\lfloor n/2\rfloor+17)+n$ 的解为 $O(n\lg n)$。

有个+17，直接证无法由$T(\frac{n}{2})\leq c\frac{n}{2}\log\frac{n}{2}$换掉$T(\frac{n}{2}+17)$。下面通过变量代换将+17抹掉：

记$n=m+34$，则$T(m+34)=2T(m/2+34)+m+34$

记$S(m)=T(m+34)$，则$S(m)=2S(m/2)+m+34$，$S(m)+34=2[S(m/2)+34]+m$

记$R(m)=S(m)+34$，则$R(m)=2R(m/2)+m$

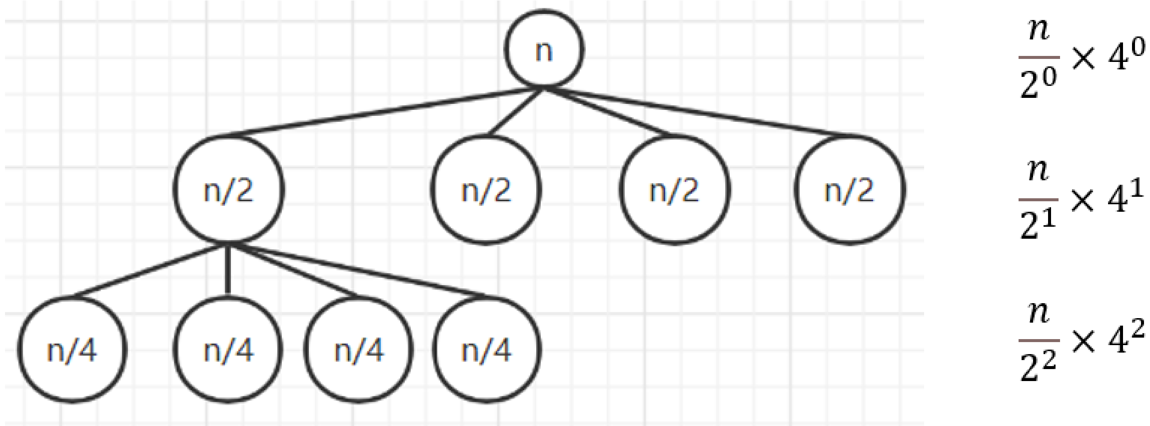
最后记得换回去：由$R(m)=O(n\log n)$，$R(m)=S(m)+34=T(m+34)+34=O(n\log n)$，则$T(n)=O(n\log n)$。

4.4

P53 4.4：3，6

4.4-3 对递归式 $T(n)=4T(n/2+2)+n$，利用递归树确定一个好的渐近上界，用代入法进行验证。

法1 先做变量替换，$T(n+4)=4T(\frac{n+4}{2})+n+4$，$S(n)=4S(\frac{n}{2})+n+4$，$S(n)+\frac{4}{3}=4[S(\frac{n}{2})+\frac{4}{3}]+n$，$R(n)=4R(\frac{n}{2})+n$，这个递归树就很好画了，



求层数：$\frac{n}{2^k}\le 1$，$k\ge \log_2 n$

求递归树的总代价：$\Sigma_{i=0}^{k}\frac{n}{2^i}4^i = n\Sigma 2^i = n\frac{2^0-2^{k+1}}{1-2} = n(2^{k+1}-1) < n2^k = n^2$，故$R(n)=O(n^2)$。因为$R(n)=\frac{3}{4}T(n+4)$，所以$T(n)=O(n^2)$。

代入法证明$R(n)=O(n^2)$：先假设$R(n)\le cn^2$在$n/2$时成立，目标证$R(n)\le cn^2$。

由递推式，$R(n)=4R(\frac{n}{2})+n\le 4c\frac{n^2}{4}+n=cn^2+n$，多了一个+n。

故要缩小假设的界，重新假设$R(n)\le cn^2-an$在n/2时成立，目标证$R(n)\le cn^2-an.$

由递推式，$R(n)\le 4(c\frac{n^2}{4}-a\frac{n}{2})+n=cn^2-2an+n$，只要这个界比要目标界紧即可，即$-2an+n\le -an$，$a\ge 1$，得证。

> 其实由主定理，$a=4,b=2,\log_b a=2$，$f(n)=n=O(n^{2-\varepsilon}),\varepsilon=1$，直接可得出$R(n)=\theta(n^2)$。

法2 直接画

层数	节点代价	节点数
0	n	4^0
1	$\frac{n}{2}+2=\frac{n}{2^2}+2$	4^1
2	$\frac{\frac{n}{2^2}+3}{2}+2=\frac{n}{2^3}+\frac{1}{2^0}+2$	4^2
3	$\frac{\frac{n}{2^3}+\frac{1}{2^0}+2}{2}+2=\frac{n}{2^4}+\frac{1}{2^1}+\frac{1}{2^0}+2$	4^3
i	$\frac{n}{2^i}+\frac{1}{2^{i-2}}+\frac{1}{2^{i-3}}+\cdots+\frac{1}{2^0}+2$	4^i

设总共有k层，则$\frac{n}{2^k}+\frac{1}{2^{k-2}}+\frac{1}{2^{k-3}}+\cdots+\frac{1}{2^0}+2\le 1$，

$\frac{n}{2^k}+\frac{2-\frac{1}{2^{k-1}}}{1-\frac{1}{2}}=\frac{n-4}{2^k}+4\le 1$，$2^k\le -\frac{n-4}{3}<0$，不对，所以最后一层不可能$\le 1$，

要保证划分后>0，最后一层最小≤ 5，$\frac{n-4}{2^k}+4\le 5$，

所以，$k\ge \log_2(n-4)$。

假设最后一层达到5，floor($\frac{5}{2}$)+2=4，继续划分$\frac{4}{2}$+2=4，即$T(4)=4T(4)+4$, $T(4)=-\frac{4}{3}$，所以$T(4)=0$，进而$T(5)=4T(4)+5=5$，最后一层最小划到5。

求总代价:

第i层行代价：$(\frac{n}{2^i}+\frac{1}{2^{i-2}}+\frac{1}{2^{i-3}}+\cdots+\frac{1}{2^0}+2)\times 4^i=n\cdot 2^i+4\cdot 4^i-4\cdot 2^i$，

k个层求和：$\Sigma_{i=0}^{k}(n\cdot 2^i+4\cdot 4^i-4\cdot 2^i)=(n-4)(2^{k+1}-1)+\frac{4}{3}(4^{k+1}-1)$

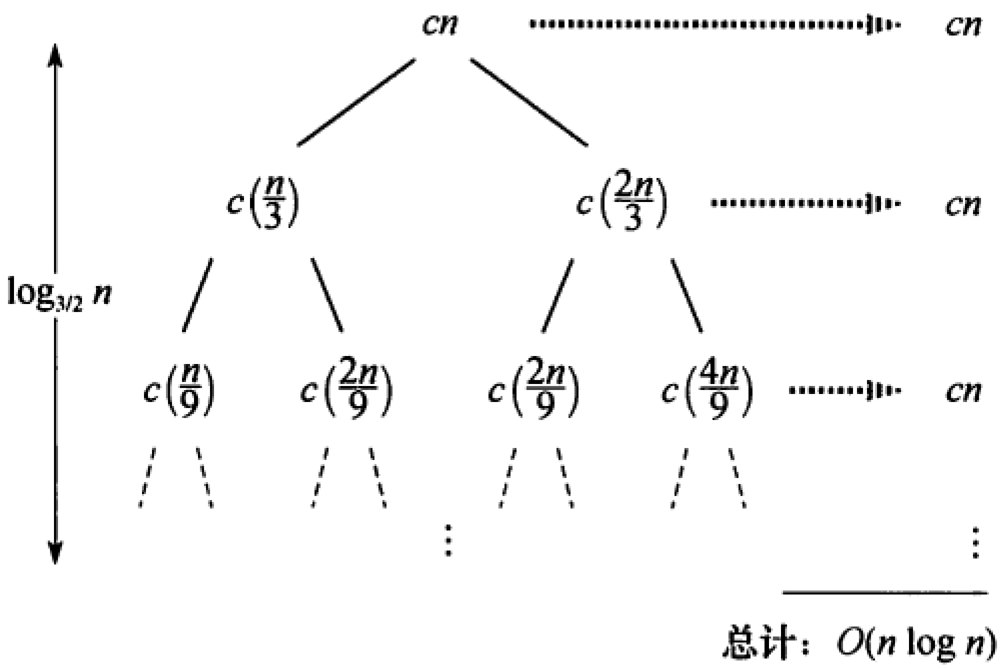
$=(n-4)[2(n-4)-1]+\frac{4}{3}[4(n-4)^2-1]=\frac{22}{3}(n-4)^2-(n-4)-\frac{4}{3}$

$=\Theta(n^2)$.

这里只要给个渐进上界，$O(n^2)$即可。

代入法证明：变量替换，转为证$R(n)=\frac{3}{4}T(n+4)=O(n^2)$。

4.4-6 对递归式 $T(n)=T(n/3)+T(2n/3)+cn$，利用递归树论证其解为 $\Omega(n\lg n)$，其中 c 为常数。



$\frac{n}{3}$的分支最早结束，其层数 $\frac{n}{3^k}\leq 1, k\geq\log_3 n$，将前k层的代价求和得下界Ω，$cn\log_3 n=\Omega(n\log n)$。

> $\frac{2n}{3}$分支结束最晚，层数=$\log_{3/2} n$，代价上界也是$O(n\log n)$。总之，$T(n)=\theta(n\log n)$。

4.5

P55 4.5：1，4

4.5-1 对下列递归式，使用主方法求出渐近紧确界。

a. $T(n)=2T(n/4)+1$

b. $T(n)=2T(n/4)+\sqrt{n}$

c. $T(n)=2T(n/4)+n$

d. $T(n)=2T(n/4)+n^2$

四题形式都符合主定理，$a=2, b=4, \log_b a=1/2$。

a. $f(n)=1=O(n^{1/2-\varepsilon})$，$\varepsilon=1/2$，符合主定理case1，$T(n)=\theta(n^{1/2})$

b. $f(n)=\sqrt{n}=\theta(n^{1/2})$，case2，$T(n)=\theta(n^{1/2}\lg n)$

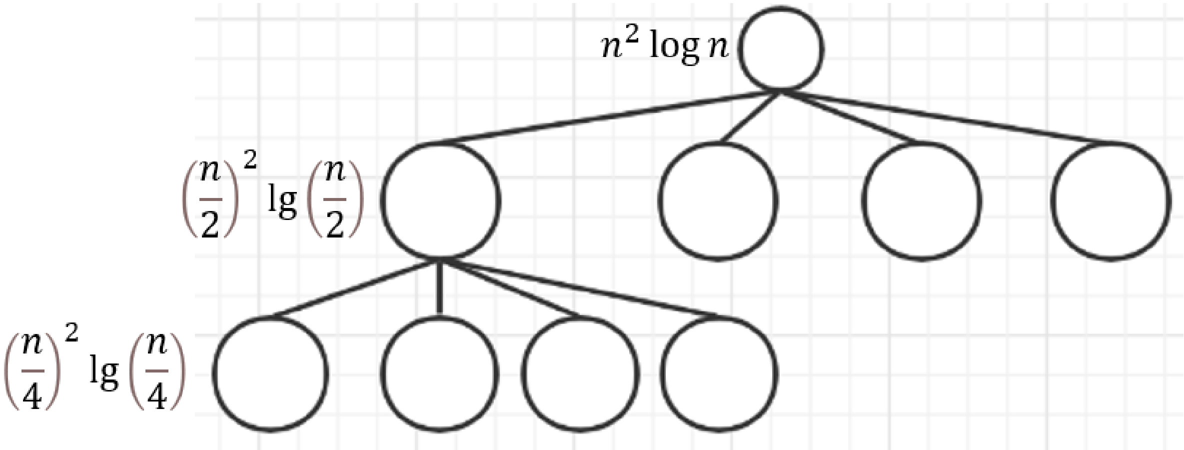
c. $f(n)=n=\Omega(n^{1/2+\varepsilon})$，$\varepsilon\in(0,1/2]$，且$2f(n/4)=n/2\le cf(n)=cn$，$c\in[1/2,1)$，case3，$T(n)=\theta(n)$

d. $f(n)=n^2=\Omega(n^{1/2+\varepsilon})$，$\varepsilon=(0,3/2]$，且$2f(n/4)=n^2/8\le cf(n)=cn^2$，$c\in[1/8,1)$，case3，$T(n)=\theta(n^2)$

4.5-4 主方法能应用于递归式 $T(n)=4T(n/2)+n^2\lg n$ 吗？请说明为什么可以或者为什么不可以。给出这个递归式的一个渐近上界。

$a=4, b=2, \log_b a=2, f(n)=n^2\lg n$渐进大于$n^{\log_b a}$，不符合主定理case2的原始形式，考虑case3，假设$f(n)=\Omega(n^{2+\varepsilon})$，则$\lg n/n^{\varepsilon}\to\infty$，但这个极限$\to 0$，故不符合case3。这个例子处于case2-3的间隙。

用递归树求界：



总层数$k=\log_2 n$，总代价

$\Sigma_{i=0}^{k}(\frac{n}{2^i})^2\log(\frac{n}{2^i})\cdot 4^i=\Sigma n^2(\lg n-i)=\Sigma n^2\lg n-\Sigma n^2 i=n^2\lg nk-n^2\frac{(0+k)(k+1)}{2}=\theta(n^2\lg^2 n)$。

其实，$f(n)=\theta(n^2\lg n)$是符合主定理case2的扩展形式的，直接可以得出$T(n)=\theta(n^2\log^2 n)$。

> 主定理case2扩展形式：若$f(n)=\theta(n^{\log_b a}\log^k n)$，则$T(n)=\theta(n^{\log_b a}\log^{k+1} n)$

！！！考试时要写不符合case2原始形式、不符合case3的分析。

6.1

P85 6.1：3，5

6.1-3 证明：在最大堆的任一子树中，该子树所包含的最大元素在该子树的根结点上。

∀子树T，从叶节点开始，由最大堆定义Parent≥Children，T所有节点一定≤ 根。

反证法：

Assume the claim is false—i.e., that there is a subtree whose root is not the largest element in the subtree. Then the maximum element is somewhere else in the subtree, possibly even at more than one location. Let m be the index at which the maximum appears (the lowest such index if the maximum appears more than once). Since the maximum is not at the root of the subtree, node m has a parent. Since the parent of a node has a lower index than the node, and m was chosen to be the smallest index of the maximum value, $A[\text{PARENT}(m)] < A[m]$. But by the max-heap property, we must have $A[\text{PARENT}(m)] \geq A[m]$. So our assumption is false, and the claim is true.

6.1-5 一个已排好序的数组是一个最小堆吗？

考虑一个升序序列A[1..n]，因为升序，$A[i] \leq A[2i]$，$A[i] \leq A[2i+1]$，$2i+1 \leq n$，即Parent≤左右孩子，就是一个最小堆。

6.2

P87 6.2：1，4

6.2-1 参照图 6-2 的方法，说明 MAX-HEAPIFY(A，3)在数组 A=⟨27，17，3，16，13，10，1，5，7，12，4，8，9，0⟩上的操作过程。

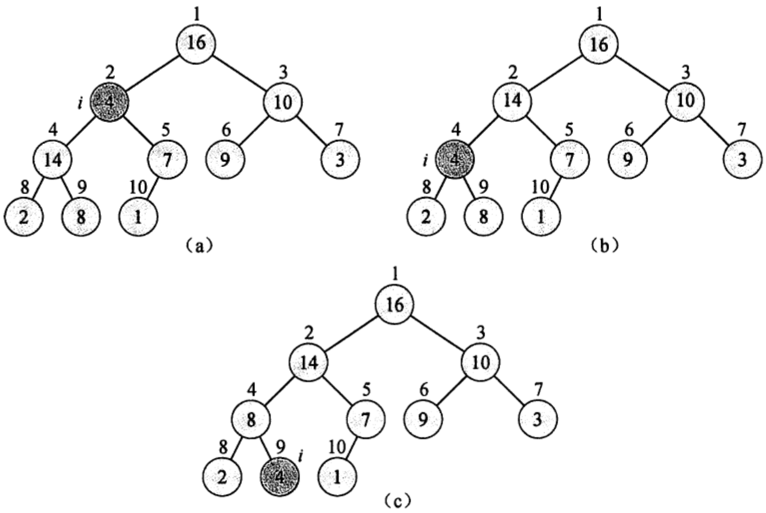
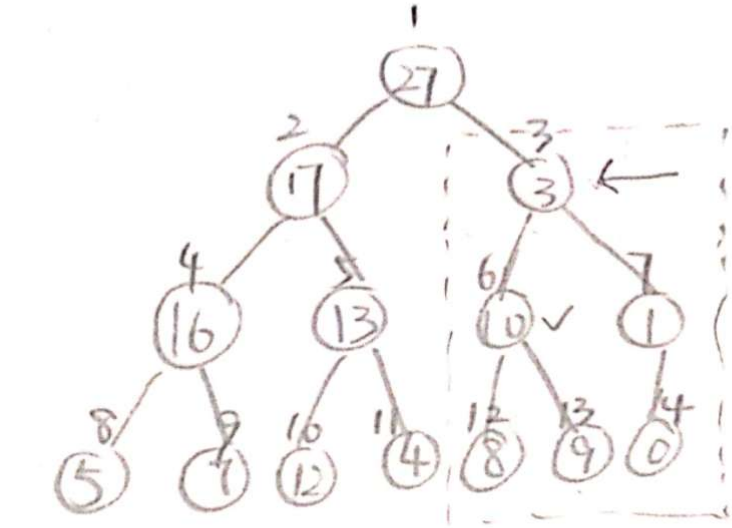
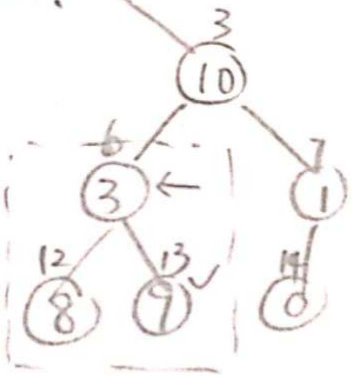


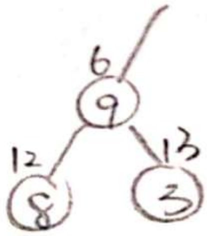
图 6-2 当 $A.heap\text{-}size$=10 时，MAX-HEAPIFY(A，2)的执行过程。(a)初始状态，在结点 i=2 处，$A[2]$违背了最大堆性质，因为它的值不大于它的孩子。在(b)中，通过交换 $A[2]$和 $A[4]$的值，结点 2 恢复了最大堆的性质，但又导致结点 4 违反了最大堆的性质。递归调用 MAX-HEAPIFY(A，4)，此时 i=4。在(c)中，通过交换 $A[4]$和 $A[9]$的值，结点 4 的最大堆性质得到了恢复。再次递归调用 MAX-HEAPIFY(A，9)，此时不再有新的数据交换



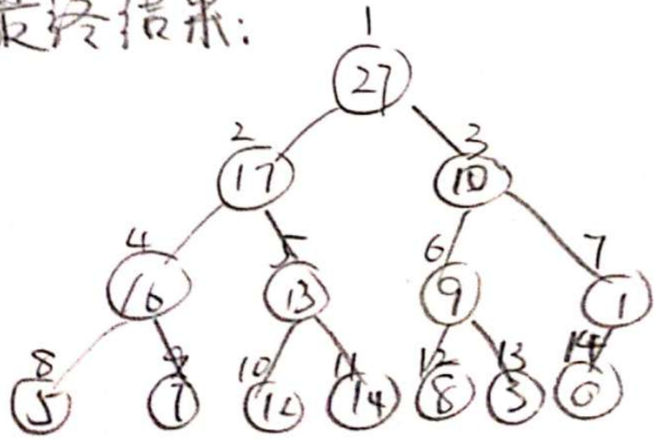
节点3的
子树符合
max-heap

MaxHeapify(A,3):





最终结果:



6.2-4 当 $i>A.heap\text{-}size/2$ 时，调用 MAX-HEAPIFY(A，i)会有什么结果？

$2i, 2i+1>$A.heapSize，A[i]无子节点，MaxHeapify(A,i) does nothing。