Homework01 2021.10.03

1.

假定 f (n) 与 g(n) 都是渐进非负函数，判断下列等式或陈述是否一定是正确的，并简要解释你的答案.
a) $f(n) = O(f(n)^2)$**.**
b)$f(n) + g(n) = Q(max(f(n), g(n)))$**.**
c) $f(n) + O(f(n)) = \Theta(f(n))$**.**
d) if$f(n) = \Omega(g(n))$**, then** $f(n) = o(g(n))$**. (注意是小 o)**

a）错误

取$f(n) = 1/n$，当n趋向于正无穷时，$\frac{f(n)}{f(n)^2} = n$趋向于正无穷，因此在此例下$f(n) = \omega(f(n)^2)$，所以原命题错误

b）正确

由$f(n) + g(n) - max(f(n), g(n)) = min(f(n), g(n)) \geq 0$可得
$max(f(n), g(n)) \leq f(n) + g(n)$
由$max(f(n) + g(n)) - \frac{f(n)+g(n)}{2} = \frac{max(f(n)+g(n)) - min(f(n)+g(n))}{2} \geq 0$可得
$max(f(n) + g(n)) \geq \frac{f(n)+g(n)}{2}$
因此$\frac{f(n)+g(n)}{2} \leq max(f(n), g(n)) \leq f(n) + g(n)$，所以$f(n) + g(n) = Q(max(f(n), g(n)))$

c）正确

由O记号的定义可知，存在正整数n_0，对于任意的n大于n_0，都有实数c，使得$0 \leq O(f(n)) \leq cf(n)$，因此$f(n) \leq f(n) + O(f(n)) \leq f(n) + cf(n) = (c+1)f(n)$，即
$f(n) + O(f(n)) = \Theta(f(n))$

d）错误

取$f(n) = g(n) = n$，满足$f(n) = \Omega(g(n))$，但是显然不满足$f(n) = o(g(n))$

2.

时间复杂度
a) 证明 $lg(n!) = \Theta(nlg(n))$**(课本等式 3:19), 并证明** $n! = w(2n)$ **且** $n! = o(n^n)$**.**
b) 使用代入法证明 $T(n) = T(\lceil n/2 \rceil) + 1$ **的解为** $O(lgn)$**.**
c) 对递归式 $T(n) = T(n-a) + T(a) + cn$**, 利用递归树给出一个渐进紧确解, 其中** $a \geq 1$ **和** $c > 0$ **为常数.**
d) 主方法能应用于递归式 $T(n) = 4T(n/2) + n^2lgn$ **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进上界**

a）由$\lg(n!) = \sum_{k=1}^{n}(\lg k) \leq \sum_{k=1}^{n}(\lg n) = n\lg n$可得$\lg(n!) = O(n\lg n)$

又由$\lg(n!) = \sum_{k=\lfloor n/2 \rfloor}^{n}(\lg k) \geq \sum_{k=\lfloor n/2 \rfloor}^{n}(\lg\lfloor n/2 \rfloor) \geq \frac{n}{2}\lg\lfloor n/2 \rfloor = \Theta(n\lg n)$可得
$\lg(n!) = \Omega(n\lg n)$

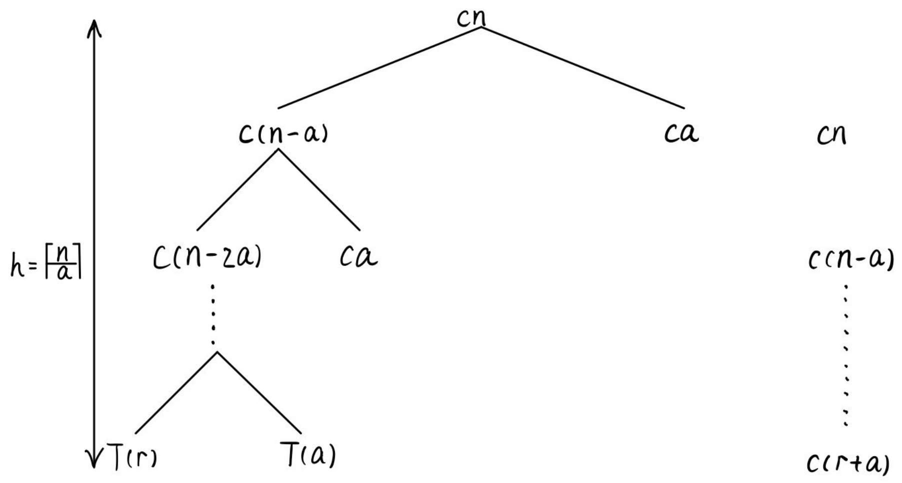
综上可得，$\lg(n!) = \Omega(n\lg n)$

b）为证$T(n) = T(\lceil n/2 \rceil) + 1 = O(\lg n)$，需要证明$T(n) \leq c\lg n$

代入可得$T(n) = T(\lceil n/2 \rceil) + 1 \leq c\lg\lceil n/2 \rceil + 1 \leq c\lg n - (c\lg 3/2 - 1)$

取$c = \frac{1}{3}$，则$c\lg 3/2 - 1 < 0$，则$T(n) \leq c\lg n$，即$T(n) = O(\lg n)$

c）递归树如图所示



$$
\begin{aligned}
T(n) = (h-1)ca + T(r) + \sum_{k=0}^{h-2}(c(n-ka)) &= (h-1)ca + dr + cn(h-1) - a\sum_{k=0}^{h-2}(k) \\
&= (\lceil n/a\rceil - 1)ca + dr + cn(\lceil n/a\rceil - 1) - \frac{(\lceil n/a\rceil - 1)(\lceil n/a\rceil - 2)}{2} \\
&= \Theta(n^2)
\end{aligned}
$$

d）不能应用

$a=4, b=2, f(n)=n^2\lg n$，因此$n^{log_b a} = n^{log_n 4} = \Theta(n^2)$，注意到尽管$n^2\lg n = \Omega(n^2)$，但是对于$\lg n = o(n^{\varepsilon})$，因此不存在$\varepsilon > 0$，使得$n^2\lg n = O(n^{2-\varepsilon})$或$n^2\lg n = \Omega(n^{2+\varepsilon})$，不满足使用主方法的条件

使用代入法，猜测$T(n)$的一个渐近上界是$n^2\lg^2 n$，只用证明$T(n) \leq cn^2\lg^2 n$

$ T(n) = 4T(n/2) + n^2 \lg n \leq c^2n^2 \lg ^2 (n/2) + n^2 \lg n \\= c^2n^2 \lg ^2 n + c^2n^2 + (1-2c^2)n^2 \lg n \leq c^2n^2 \lg ^2 n + (1-c^2)n^2 \lg n $

取$c=1$，则$T(n) \leq cn^2\lg^2 n$，因此$T(n) = O(n^2\lg^2 n)$

3.

对下列递归式, 使用主方法求出渐进紧确解:
a) $T(n) = 2T(n/4) + \sqrt{n}$
b) $T(n) = 2T(n/4) + n^2$.

a）$a=2, b=4, f(n)=\sqrt{n}$，因此$n^{log_b a} = n^{log_4 2} = \Theta(\sqrt{n})$，由于$f(n) = \Theta(n^{log_4 2}) = \Theta(\sqrt{n})$，因此可以应用主定理的情况2，从而得到解$T(n) = \Theta(\sqrt{n}\lg n)$
b）$a=2, b=4, f(n)=n^2$，因此$n^{log_b a} = n^{log_4 2} = \Theta(\sqrt{n})$，由于 $f(n) = \Omega(n^{log_4 2+\varepsilon})$，其中$\varepsilon = 1.5$，因此可以应用主定理的情况3，从而得到解$T(n) = \Theta(n^2)$

4.

考虑以下查找问题:
输入: n 个数的一个序列 $A = a_1, a_2, , , , , , , a_n$ 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.
a) 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 Loop Invariant (循环不变式) 来证明你的算法是正确的.
b) 假定 v 等可能的为数组中的任意元素，平均需要检查序列的多少元素？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运行时间.

a）伪代码如下所示：

```
for i = 1 to A.length do
    if v == A[i] then
        return i
return NIL
```

初始化：算法开始时$i = 1$，还没有开始查找序列，故v不在已查找的序列中

保持：若在此次（第i次）查找之前找到v，则返回$i-1$的值；若没有找到v，则执行第i查找。下一次的查找的情况同此次查找，对于算法而言都为真

终止：循环终止时，若找到v，则在第i次查找是已返回i的值；若没有找到v，则返回NIL.说明算法是正确的

b）算法的分析如下

- 平均情况：v在序列中各位置的概率均为$1/n$，因此平均检查次数为$1/n\sum_{k=1}^{n} n = \frac{n+1}{2} = \Theta(\frac{n}{2})$
- 最坏情况：v是序列的最后一个元素或v不在序列中，此时需要检查$\Theta(n)$次

5.

堆排序：
对于一个按升序排列的包含 n 个元素的有序数组 A 来说， Heapsort 的时间复杂度是多少？如果 A是降序的呢？请简要分析并给出结果.

堆排序分为两步，即初始化堆、调整堆。
设元素个数为n，则堆的高度$k = \lg(n+1) \approx \lg n$，非叶子结点的个数为$2^{(k-1)} - 1$

- 升序序列：建堆时每个非叶子结点都需要进行调整，则第i层的非叶子结点需要的操作次数为k-i，第i层共有$2^{(i-1)}$个结点，则第i层的所有结点所做的操作为$k \times 2^{i-1} - i \times 2^{i-1}$，

  共k-1层非叶子结点，总的操作次数为
  $\sum_{i=1}^{k-1}(k \times 2^{i-1} - i \times 2^{i-1}) = 2^k - k + 1 = n - \lg n + 1 = O(n)$
  排序时，根节点和排在最后的序号为m的叶子结点交换，并进行调整，那么调整的操作次数 = 原来m结点所在的层数 = 堆的高度（因为m结点在堆的最后）=$\lg m$

  所以总的操作次数为$\sum_{m=1}^{n-1}\lg m = \lg((n-1)!) = O(n \lg n)$

  因此$T(n) = O(n) + O(n \lg n) = O(n \lg n)$
- 降序序列：建堆时序列即为大根堆，只需要比较不需要调整位置，排序与升序同理

  所以$T(n) = O(n/2) + O(n \lg n) = O(n \lg n)$

6.

快速排序：

1. **假设快速排序的每一层所做的划分比例都是 1 −a : a，其中 0 < a ≤ 1/2 且是一个常数. 试证明：在相应的递归树中，叶结点的最小深度大约是 $-lgn/lga$，最大深度大约是 $-lgn/lg(1-a)$ (无需考虑舍入问题).**
2. **试证明：在一个随机输入数组上，对于任何常数 0 < a ≤ 1/2， Partition 产生比 1 −a : a 更平衡的划分的概率约为 1 − 2a.**

1. 假设在递归树中，结点左孩子的问题规模均为其父节点的$1-a$倍，右孩子的问题规模均为其父节点的a倍，则从根结点沿左孩子达到的叶子节点是深度最大的结点，深度为

$\log_{\frac{1}{1-a}} n = -\lg n / \lg(1-a)$；从根结点沿右孩子达到的叶子节点是深度最小的结点，深度为 $\log_{\frac{1}{a}} n = -\lg n / \lg a$

2. 最理想的情况是Partition算法产生的划分比例刚好是$1/2 : 1/2$，则Partition 产生比 1 −a : a 更平衡的划分的概率约为$\frac{1/2-a}{1/2} = 1-2a$