HW1

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) $f(n)=O\left(f(n)^2\right)$ 错误，考虑$f(n)=1/n$
   b) $f(n)+g(n)=\Theta(\max(f(n),g(n)))$ 正确
   $\max(f(n),g(n))<f(n)+g(n)<2\max(f(n),g(n))$
   c)$f(n)+O(f(n))=\Theta(f(n))$ 正确，令$g(n)=O(f(n))$，这就是b)的一个特殊情况
   d)if $f(n)=\Omega(g(n))$, then $g(n)=o(f(n))$. (注意是小 o) 错误 考虑$f(n)=g(n)$

2. 时间复杂度
   a) 证明 $\lg(n!)=\Theta(n\lg(n))$(课本等式 3.19), 并证明 $n!=\omega\left(2^n\right)$ 且 $n!=o\left(n^n\right)$.
   上界：$\lg(n!)=\sum_{i=1}^{n}\lg i<=\sum_{i=1}^{n}\lg n=nlgn$
   下界：$\lg(n!)=\sum_{i=1}^{n}\lg i>\sum_{i=n/2}^{n}\lg i>n/2*\lg(n/2)=n/2*\lg n-n/2*\lg 2$
   我们只算了他的后一半，并且后一半是按照最低的来计算

   这个也可以用stirling公式来证明
   由ω和o的定义，看他们比值的极限：
   $\lim_{n\to\infty}\frac{2^n}{n!}=0\to n!=\omega\left(2^n\right)$
   $\lim_{n\to\infty}\frac{n^n}{n!}=\infty\to n!=o\left(n^n\right)$
   b) 使用代人法证明 $T(n)=T(\lceil n/2\rceil)+1$ 的解为 $O(\mathrm{lgn})$.
   代入$T(n)\leq 3\log n-1$ 或者$c\lg n$
   $$\begin{aligned}T(n)&=T(\lceil n/2\rceil)+1\\&\leq 3\log\left(\lceil n/2\rceil\right)-1+1\\&\leq 3\log\left(3n/4\right)\\&=3\log n+3\log\left(3/4\right)\\&\leq 3\log n+\log\left(1/2\right)\\&=3\log n-1\end{aligned}$$
   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 ≥ 1 和 c > 0 为常数
   T(a)可以当作常数处理，深度是n/a，$T(n)=\Theta\left(n^2\right)$

   $$T(a)+cn$$
   $$T(a)+c(n-a)$$
   $$T(a)+c(n-2a)$$
   $$T(1)$$

   d) 主方法能应用于递归式 $T(n)=4T(n/2)+n^2\,\mathrm{lgn}$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.

不可以，主方法的定义如下：

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \le cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

这里$n^{\log_b a} = n^2$,f(n)比较大，但是$f(n)/n^2 = \lg n$ 渐进小于 n^ϵ ，所以f(n)不是多项式的大于，不能用主方法求解。
$T(n) = O\left(n^2 \lg^2 n\right)$ 可以画递归树求解，深度是lgn

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) $T(n) = 2T(n/4) + n^2$
   这里$n^{\log_b a} = \sqrt{n}$,对于a)来说，就是主方法的第二种情况，$\Theta(\sqrt{n}\lg(n))$,对于b)来说是第三种，$\Theta\left(n^2\right)$
4. 考虑以下查找问题:
   输入: n 个数的一个序列 $A = a_1, a_2, \ldots, a_n$ 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.

```
def LinearSearch(v, A):
    for i in range(len(A)):
        if A[i]==v: return i
return NIL
```

   循环不变式的证明：

   初始化：在第一次迭代前子数组B[]为空，里面不包含值v

   保持：每次迭代，会判断一个新的值，如果新增加的值为v，那么循环终止，如果不终止的话，加入子数组，子数组中也不会包含值v

   终止：终止的时候便利了所有的元素，该元素中不包含值v
   b) 平均情况: $\Theta(n)$ 最坏运行: $\Theta(n)$
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.
   无论数组为升序或者降序，建成一个堆的时间复杂度均为 $O(n)$。
   在排序过程中需要不断维护最大堆的性质，共调用 $n-1$ 次 Maxheap，每次调用时间复杂度为 $O(\log n)$， $n-1$ 次调用时间复杂度为 $O(n\log n)$。
   综上，无论数组为升序或者降序，Heapsort 的时间复杂度均为 $O(n\log n)$。
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$1-\alpha:\alpha$, 其中 $0 < \alpha \le 1/2$ 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$-\lg n/\lg\alpha$ 最大深度大约是 $-\lg n/\lg(1-\alpha)$
      $T(n) = T(n/\alpha) + T(n/(1-\alpha))$
      最小深度：$n\alpha^x \approx 1 \Rightarrow x \ge -\log_\alpha n = -\frac{\lg n}{\lg\alpha}$
      最大深度：$n(1-\alpha)^x \approx 1 \Rightarrow x \ge -\log_\alpha n = -\frac{\lg n}{\lg(1-\alpha)}$

2. 试证明：在一个随机输入数组上，对于任何常数$0 < \alpha \leq 1/2$,Partition 产生比$1-\alpha:\alpha$ 更平衡的划分的概率约为$1-2\alpha$.
   设数组为 A，不失一般性，设其所有元素两两不同。那么这时只需考虑其元素的个数，不妨假设其为1 到 n 这 n 个整数的随机排列。如果要更平衡，那就是我们选中的那个数值k需要满足$\alpha n < k < (1-\alpha)n$，因为是随机排序，所以，这个的概率是$1-2\alpha$。

HW2

1. 下面的排序算法中哪些是稳定的：插入排序、归并排序、堆排序、快速排序和计数排序？给出一 能使任何排序算法都稳定的方法。你所给出的方法带来的额外时间和空间开销是多少？
   稳定排序：插入排序、归并排序、计数排序。
   不稳定排序：堆排序、快速排序。
   方法示例：将数组中的元素替换为包含元素本身及其索引的一个数对，在比较的时候，先比较元素大小， 若元素大小相同则比较索引。
   时空开销：空间开销翻倍，（渐进）时间复杂度不变。
2. 假设所有元素都是互异的，说明在最坏情况下,如何使快速排序的运行时间为 $O(nlogn)$。
   快排的最坏情况就是划分的不平衡，如果划分出来是1和n-1的情况，快排的worst case就是$O(n^2)$了,想要改进快排，那就是让我们的划分更加均匀，找到一个中位数，median 可以线性时间查找到。
   $T(n)=2T(n/2)+O(n)$,主方法带入，$O(nlogn)$
3. 给定一个整数数组，其中不同的整数所包含的数字的位数可能不同。但该数组中，所有整数中包含的总数字位数为n。设计算法使其可以在$O(n)$时间内对该数组进行排序。
   先把每个元素按照他们的位数划分，遍历一遍数组，这个过程的复杂度是$O(n)$
   对于各组的元素，我们基数排序，对于第i组来说，假设有l_i个数，每个数有m_i位，基数排序中嵌套计数排序，那么每组基数排序的时间是$O(m_i * l_i)$,把每个组的值加在一起
   $\sum_i O(m_i * l_i) = O(n)$
4. SELECT 算法最坏情况下的比较次数 $T(n)=\Theta(n)$，但是其中的常数项使非常大的。请对其进行优化，使其满足：
   - 在最坏情况下的比较次数为 $\Theta(n)$
   - 当i 是小于 n/2 的常数时，最坏情况下只需要进行 $n+O(logn)$ 次比较

   如果i 是大于 n/2 的常数，那么直接使用原始版本，如果i 小于 n/2 ，把他们两两分组比较，得到较小的那个分组，这里比较了n/2次。
   对于这n/2个数，我们调用自身S(n/2,i)，找他的第i小的元素m，现在，以m来划分，第i小的元素只能在前面的i组中产生，也就是前2i个元素里，找到的时间T(2i).
   $U(n,i)=U(n/2,i)+n/2+T(2i)$