作业3解答：

1. 解以下递归方程，其中 $T(1)=1$，对于 $n\geq 2$，$T(n)$ 满足以下式子：

(a) $T(n)=3T(n/2)+n$

(b) $T(n)=3T(n/2)+n^2$

(c) $T(n)=8T(n/2)+n^3$

给出 $T(n)$ 的 Ω 和 O 估计。

解：(a) $a=3$，$b=2$，$d(n)=n=O(n^{\log_b a-\varepsilon})=O(n^{\log_2 3-\varepsilon})$
$=O(n^{1.59-0.01})$，其中，$\varepsilon=0.01$。

由通解定理，$T(n)=\Theta(n^{\log_b a})=\Theta(n^{1.59})$

从而 $T(n)=O(n^{1.59})$，$T(n)=\Omega(n^{1.59})$。

(b) $a=3$，$b=2$，$d(n)=n^2=\Omega(n^{\log_b a+\varepsilon})=\Omega(n^{\log_2 3+\varepsilon})$
$=\Omega(n^{1.59+0.1})$，其中 $\varepsilon=0.1$。

又因为存在常数 $c=\frac{3}{4}<1$，
$ad(n/b)=3(\frac{n}{2})^2\leq\frac{3}{4}n^2=\frac{3}{4}d(n)$，对所有充分大的 n 成立。

由通解定理，$T(n)=\Theta(d(n))=\Theta(n^2)$，

从而 $T(n)=O(n^2)$ 且 $T(n)=\Omega(n^2)$。

(c) $a=8$，$b=2$，$d(n)=n^3=\Theta(n^{\log_b a})=\Theta(n^{\log_2 8})=\Theta(n^3)$。

由通解定理，$T(n)=\Theta(n^{\log_b a}\log_b n)=\Theta(n^3\log n)$。

故 $T(n)=O(n^3\log n)$，且 $T(n)=\Omega(n^3\log n)$。

2. 给出以下递归方程的 $T(n)$ 的 Ω 和 O 估计。

$$T(1)=1$$

$$T(n)=2T(n/2)+\log n。$$

2 解：$a=2$，$b=2$，$d(n)=\log n=O(n^{\log_b a-\varepsilon})=O(n^{1-\varepsilon})$
$=O(n^{1-0.1})$，其中 $\varepsilon=0.1$。

由通解定理，$T(n)=\Theta(n^{\log_b a})=\Theta(n^{\log_2 2})=\Theta(n)$。

从而 $T(n)=O(n)$ 且 $T(n)=\Omega(n)$。

3. 对n个Merge-Find操作问题设计一个$O(n\log\log n)$的算法。使用这样一棵树作数据结构，它的所有叶子与树根的距离为2，限制树根的度为1到$n/\log n$之间，限制树根的每一个儿子的度为1到$\log n$之间。

3.解：我们非形式化地描述这个算法，并作算法分析。我们使用这样一棵树来代表一个集合，它的所有叶子与树根r的距离为2，限制树根的度为1到$n/\log n$之间，限制树根的每个儿子S_i的度为1到$\log n$之间。树根r中保存集合名和本集合元素个数信息。r的每个儿子S_i保存该子树中叶子的个数信息。该树的每个叶子 保存该集合的一个元素。注意每一棵树除了一个儿子S_r外，其它儿子S_i的度(叶子个数)都等于$\log n$ ($i=1,2,\cdots,r-1$)。我们称S_i ($1\le i\le r-1$)是满的。而S_r有可能不满。

每一个集合A对应这样一棵树。每一个集合元素通过一个映射，映射到一棵树(它所属的集合)的一个叶子。以下是MFSET的操作和时间复杂性分析：

- INITIAL(x, A)：建立一棵树，树根标记为A，它只有一个儿子S，而S只有一个儿子(叶子)，标记为x。
- FIND(x)：通过映射找到x所在的集合A的那棵树的叶子，经过指向父结点的指针找到它所属的儿子，儿子又有指针指向树根，从而找到树根，得到集合名A。因为叶子到根距离为2，这只需O(1)时间。
- MERGE(A, B)：通过A, B名字找到它们对应的树，根据A和B的元素个数，将元素个数小的那个集合合

中山大学本科生考试答题纸

学院(系)　　专业　　级

考试科目　　成绩评定

考生姓名　　教师签名

学　号　　年　月　日

警示 《中山大学授予学士学位工作细则》第八条："考试作弊者不授予学士学位。"

并到元素个数大的那个集合中。设A树对应的儿子为$s_1, s_2, \cdots, s_k$；B树对应的儿子为$t_1, t_2, \cdots, t_m$；其中s_k和t_m可能是不满的，而其它s_i和t_j是满的($1 \le i < k$, $1 \le j < m$)。不失一般性，设$|A| \ge |B|$。我们的做法是：将$t_1, t_2, \cdots, t_{m-1}$作为A的儿子。再判定$s_k$和$t_m$哪个叶子数少。将叶子少的那个儿子的叶子加入到叶子多的那个儿子中。若加满了(即有$\log n$个叶子了)，将剩下的叶子及它们所属的儿子作为A树的儿子。刚才加满的那个儿子如果是t_m，则将它作为儿子加入A树。

初始时，每个集合A只有一个儿子，儿子只有一个叶子。作$n-1$次Merge。因为每次Merge，集合的元素个数比参加Merge的两个集合中元素少的那个集合多了一倍。至多经过$\log n$次Merge，就能把所有元素合并成一个集合。

在做Merge(A,B)时，我们做了$\lfloor |B|/\log n \rfloor$次把B的儿子加入A的操作，还做了$s_k$和$t_m$合并操作。每次合并，设儿子的叶子数比$s_k$和$t_m$中数叶子数少的那个儿子的叶子数多一倍。至多经过$\log\log n$次合并，合并后的儿子就满了。故整棵树叶子移动的次数为$n \log\log n$次。

而儿子移动的次数如下：整棵树共有个数为 $n/\log n$，每次把儿子数少的那棵树的儿子移到儿子数多的那棵树上，合并后的树的儿子数比合并前儿子数少的那棵树至少增加一倍。故每个儿子至多变换 $\log(n/\log n)$ 次所属的树。所有儿子共变换 $(n/\log n)\log(n/\log n)$ 次所属的树。故总的变换次数（即时间复杂度为）：$O((n/\log n)\log(n/\log n)+n\log\log n)=O((n/\log n)(\log n-\log\log n)+n\log\log n)=O(n-(n/\log n)\log\log n+n\log\log n)=O(n\log\log n)$。