Homework 8 问题复杂度分析

教材习题： 8.1，8.2，8.4，8.7，8.10

8.1

取比较为基本运算

1

对于G1是否为G2的补图，等价于每两个点间的所有边，要么属于G1，要么属于G2，所以需要比较的次数为$O(n^2)$
更确切的设计算法：
对于输入的邻接矩阵，由于是简单图，所以对角线上的元素都为0。
此时无论图是有向图还是无向图，其邻接矩阵都是对称的或反对称的，因此只需要比较不包括对角线的上三角矩阵的元素即可。
所以只需要比较不包括对角线的上三角矩阵的元素即可，比较G1，G2的临界矩阵内对应位置元素是否相等，只有存在任何一项相等就说明不是补图，否则是。所以比较次数为$n(n-1)/2$

2

由于问题至少需要读到G1和G2中任意两点间是否存在边，故问题的读取规模是 $n(n-1)$ ，而1次比较涉及2个值，每个值必须参与一次比较（否则修改此值会改变结果），问题的复杂度的一个下界为$n(n-1)/2$。而这个下界正好等于我们给出的可行算法的复杂度，所以这个算法是最优的，这个下界是紧的。

8.2

1

取乘为基础运算
算法：
递归的进行加乘运算。从$ans_n = an$开始，每次更新$ans_{i-1} = ans_i * x + a_{i-1}$，直到$i = 0$，此时$ans_0$即为答案。每个$a_i$恰好乘了i次x，所以算法正确。
共进行了n次乘法和n次加法，所以复杂度为$O(n)$

2

这里涉及一个问题，这个$\Omega(n)$的复杂度是基于什么运算说明的。
如果是基于加法运算，那显然每个ai都至少参与一次加法运算，所以复杂度的下界为$O(n)$,自然所有算法的复杂度都是$\Omega(n)$。
但如果是基于乘法运算讨论，那以上的说法或许不合适，更合适的说法是由于x^i对于不同的i来说线性无关，因此每个ai必须参与与其x^i的乘法，否则在x的改变下算法的结果一定错误，因此每个ai都至少参与一次乘法运算，所以复杂度的下界为$O(n)$,自然所有算法的复杂度都是$\Omega(n)$。

8.4

1

总复杂度为每次的归并复杂度累加：$\frac{n}{k}[(1+1)+(2+1)+\ldots\ldots+(k-1+1)]=\frac{n(k^2+k-2)}{2k}----O(nk)$

2

显然前一个算法的复杂度高的原因是后面每次都使最长的已有串参与了归并，故改进为每次选取已有串中最小的两串参与排序，这样可以近似的理解为先将所有的n/k长串归并为2n/k串，再将2n/k串归并为4n/k串，……，直到归并为n长的串，即一串。这样每层归并的复杂度为n，总层数为$O(log(k))$,故总复杂度$O(nlogk)$

3

直接计算在已知这些有序子串后，所有可能情况数：即总排列数除各个子串的可能排列数（相当于损失了这些自由度）：$\frac{n!}{(n/k)!^k}$
每次比较至多二分，故二叉树的深度即最坏情况下的最低比较次数，该值为

$$log(W_{自由度})=log(\frac{n!}{(n/k)!^k})=nlogn-k*n/k*(logn-logk)=nlogk$$

因此最坏情况下的复杂度下界为$O(nlogk)$。说明2中的算法为最优算法。

8.7

1

算法设计：
由于不等整数严格递增，说明每个数至少比前一个数大1，由此得两个性质：

- 若ai>i，则ai+1>i+1，则所有比i靠后的元素aj都有aj>j,不可能取等
- 若ai<i，则ai+1<i+1，则所有比i靠前的元素aj都有aj<j,不可能取等
  因此已知存在取等，故$a_0 \leq 0, a_n \geq n$，故可以二分查找，每次判断ai与i的大小关系，若ai>i，则在左半区再次二分查找，否则在右半区查找，直到找到ai=i，此时i即为答案。
  伪代码：

```
a[],l<-1,r<-n;
find(int a[],int l,int r){
    mid<-(l+r)/2;
    if(a[mid]>mid) return find(a,l,mid);
    else if a[mid]==mid return mid;
    else return find(a,mid+1,r);
}
```

复杂度：$O(logn)$

2

由于从1到n任何一个i都可能成为解，故有n种可能情况，而每次比较提供一次二分信息，则该二叉树有n个叶子节点，故深度至少为$logn$，故复杂度下界为$O(logn)$，至少进行$\Omega(logn)$次比较。

8.10

n个数的列中第k小的数a，即找到该数满足有k-1个数小于它且有n-k个数大于它。
相当于算法需要证明，输入中有k-1个数小于a，且有n-k个数大于a。
对于算法给定的一次比较,可以用赋值策略：

- 如果是比较一个已知大于a的数与新读数,则令新读的数小于a，则不能得到新的数的大小关系，故不增加信息。
- 如果是比较一个已知小于a的数与新读数,则令新读的数大于a，则不能得到新的数的大小关系，故不增加信息。
- 如果是比较两个新读的数，则令二者分开在a的两侧，因此这样的比较没有带来信息。
- 如果是比较两个已知的数，则仅当比较结果为x1>x2且已知x2>a时（或反过来）能增加一个信息，至多增加一次信息。
  因此，由于最终的问题我们需要k-1+n-k=n-1个信息，算法至少需要进行最后一种比较n-1次，但由于所有第一次读入的数，我们都可以用上面的策略自由的分配新读数的大小，使其参与的比较不增加信息。由于以上前三种策略每次至多给大和小的数池分别增加一个数，那么这种浪费的策略至少可以进行$min[k-1, n-k]$次，即直到k-1个小的数或n-k个大的数的池满了之后就不能自由的分配新读入的数了。
  因此算法至少需要进行$n-1+min[k-1, n-k]$次比较，即$n+min[k, n-k+1]-2$次比较。