A

点分治，记录每个点到根和从根走下来的答案，到根的记录最后一个元素大小，从根走下来记录第一个元素大小。

复杂度$O(nk\log n+qk)$

B

首先令$x_i=\sum_{j=1}^{i-1}[P_i<P_j]$。$x$与$P$双射，于是就可以通过$x$刻画好位置了。 那么就是每次冒泡排序后统计$x$中$>0$的数的连续段个数。 每次冒泡排序后$x$中大于0 的元素会减一然后整体前移。

可以发现序列x的贡献为$\sum\max(x_i-x_{i-1},0)$，这个东西可以拆开然后算贡献，但复杂度比较大。

进一步观察发现 $\sum\max(x_i-x_{i-1},0)$等价与$\sum v_i$， v_i表示第i个位置后面比P_i小的那些数对应的位置的连续段个数。 这个就比较好算，枚举一个位置i ，一个位置$j(i<j)$当且仅当$P_i>P_j$且$P_{j+1}>P_i$ 。 算的时候枚举填啥，然后其他数随便填，得到一个$O(n^3)$的算法。 预处理组合数，结合前缀和优化可以做到$O(n^2)$。

进一步展开组合数发现可以从前往后用BIT或者线段树维护组合系数的变化，总复杂度$O(n\log n)$

C

考虑建立左儿子右兄弟表示法。先不管如何求，假设我们求出了每个点的左儿子、右兄弟和父亲，我们该如何求出 DFS 序：

首先，维护当前所在的点。

如果这个点有左儿子我们就向它的左儿子移动，并删除这条边；如果它没有左儿子（这里可能是子树回溯上来，也可能是叶子），说明它的子树已经完成 DFS，我们将其标记为访问完成（用左儿子的数组即可，因为左儿子已经被删除，且这个点不会再被访问了），然后前往它的右兄弟。如果它没有右兄弟，则应该前往它的父亲。

从根出发开始该过程，每个点被标记的顺序就是树的后序遍历，即 DFS 序的反序。

这里需要消耗 $3n$ 的空间。但是注意到右兄弟和父亲是没有区别的，将没有右兄弟的点的右兄弟设置为父亲，就可以不需要父亲数组了。

现在就是如何求出左儿子和右兄弟。

我们按任意顺序枚举每个点去增量维护，如果它的父亲目前没有左儿子，就将左儿子设置为当前点；否则我们考虑类似链式前向星的做法，将自己的右兄弟设置为父亲的左儿子，然后将父亲的左儿子设置为自己即可。

这里的右儿子直接存在 f 数组里，这样没有右儿子的点右儿子就自动为父亲了。

于是就做完了。

```
#ifndef GREAT_DFS_LIVES_FOREVER
#define GREAT_DFS_LIVES_FOREVER
class Array {
    int a[10000001];
public:
    Array();
    int operator[](int x);
    void set(int x,int k);
};
void dfs(int,Array&,Array&);
int fa(int);
```

```
#endif
// write your code here
void dfs(int n,Array &fa,Array &a) {
    for (int i{1};i<=n;++i)
        if (a[fa[i]]==0) a.set(fa[i],i);
        else {
            int x{fa[i]};
            fa.set(i,a[x]);
            a.set(x,i);
        }
    int c{1},tm{0};
    while (tm<n) {
        if (a[c]==0) {
            a.set(c,tm++);
            c=fa[c];
        } else {
            int p{a[c]};
            a.set(c,0);
            c=p;
        }
    }
    for (int i{1};i<=n;++i) fa.set(n-a[i],i);
    for (int i{1};i<=n;++i) a.set(i,fa[i]);
}
```