A

首先某一个点对不会被到达多次，于是先二分答案，转化为最少添加几个传送锚点。

然后对于每两个点对计算需要添加多少个传送锚点，然后跑最短路即可。

B

对于 20% 的部分直接爆搜。

记 $A=\max\{a_i\}$。

对于 50% 的部分考虑到值域很小，设 `dp[i][j]` 表示表示前 i 个数，结尾是 j 的最小花费，$\mathcal{O}(nA^2)$ 转移即可。

对于 70% 的部分，思考是否有一种方法能够不用枚举前一个大小是多少。因为 c 是固定的，所以从 `dp[i-1][j]` 转移到 `dp[i][k]`，会额外花费 $c\times|j-k|$。

于是记录 `sum[i][j]` 表示 `min{dp[i][k]+c*(k-j)}`，于是 `sum[i][j]=min{sum[i][j-1]+c,dp[i][j]}`，正着反着扫一遍预处理，就可以做到 $\mathcal{O}(1)$ 转移。需要滚动数组避免空间爆炸，时间复杂度 $\mathcal{O}(nA)$。

对于所有数据，我们发现对于左右两边都比自己大的数，肯定会不停往上增加，直到 $x^2>2c$。

但在那之前，如果它与左右两边某一个数一样大了，那么两个数一定会一起向上增加或是一起停止，否则无贡献甚至负贡献。

可以发现这个 x^2 的限制很烦，考虑拆开。因为 $(x+1)^2-x^2=2x+1$，所以我们只需要维护 $\sum x$ 即可。

所以大致思路就是找到 $[l,r]$ 的最大值 `a[mid]`，然后拆成左右两个区间，递归去做。如果左右两个区间都能升到和 `a[mid]` 一样的大小，那么就把这个区间看成一个整体继续向上加，否则继续向上加是没有意义的。

由于值域是 10^6，所以向上加不能枚举加的数量，二分或者直接计算加多少都可以。

时间复杂度 $\mathcal{O}(n\log n)$。

C

20% 的部分暴力合并集合，时间复杂度是 $\mathcal{O}(n\sum c_i+n\sum k_i)$。

对于 $c_i=1$ 的部分，如果把每个集合的下标与需要交起来的的集合下标连边，发现最后会形成森林。我们建立一个根节点把他们拼成一棵树。那么集合 B_i 就是节点 i 到根节点路上所有的数组成的集合。所以每次询问我们只找到这些关键点所在的最小联通块，统计出所有点到根节点路径的并即可，这个只需要将关键点按照 dfn 排序后求所有点的深度和减去相邻两个点的 LCA 深度和。

对于 $k\le 2$ 的部分，还是把每个集合的下标与需要交起来的的集合下标连边，如果 $x\in B_i$，那么从节点 i 出发，走到终点的所有路径都要经过点 x。根据这点我们建立支配树就好了。因为 k 很小，我们可以考虑手动分讨。

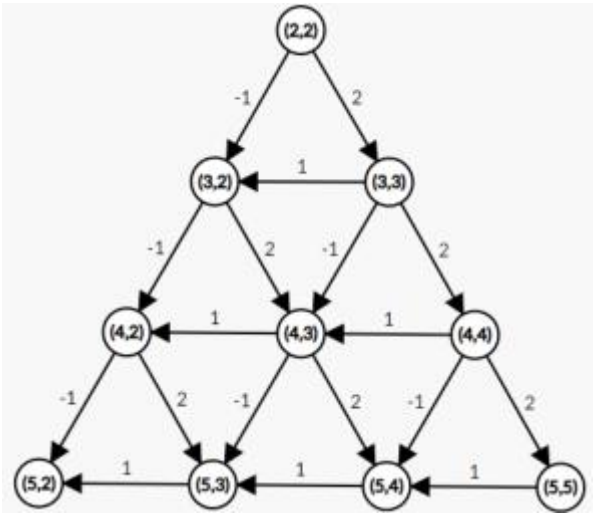
结合上面两个部分分，正解其实就是在建好的支配树上用 $c_i=1$ 的部分分做法计算就好了，时间复杂度 $\mathcal{O}(n\log n)$。

D

手玩/爆搜出 $n=4$ 时不合法的两个排列是 3142 和 4132，猜测不合法当且仅当存在这样相对大小关系的子序列，跑一下 $n=5$ 发现是对的。充分性显而易见，必要性可以将 n 缩小到 4 来证明。

考虑 dp 出合法的方案数，从大到小插入元素，一个元素插入后作为不合法的末端，当且仅当前面两个比它大的元素中间夹了一个比它小的。那么也就是插入到一个地方就保证这个位置前面已经插入的所有元素中间不能再插入元素了。设 $f_{i,j}$ 表示插了 i 个数，有 j 块（块之间可以插元素）的方案数，转移枚举插在哪个空隙，发现插在最开头和前两个块中间会使块数加一，其它的会使 j 块变为 $2\le k\le j$ 块，转移就是 $f_{i,j}\times 2\to f_{i+1,j+1}$，$f_{i,j}\to f_{i,k}(2\le k\le j)$，直接写是 $O(n^3)$ 的。优化成 $O(n^2)$ 也是容易的，可以得到 $f_{i,j}=f_{i,j+1}+2f_{i-1,j-1}-f_{i-1,j}$。

$n=1$ 是特殊的，需要特判掉。否则这个 dp 是一个路径权值计数的模型，如下图。



要求的就是 $\sum_{2\le i\le n}f_{n,i}=f_{n+1,2}$，把两维都减二，转为 $(0,0)\to(n-1,0)$ 并令 $n\leftarrow n-1$。

枚举向右下走了 i 步，贡献为 $(-1)^{n-i}2^i\binom{n+i}{n-i}C_i$，其中 C 是卡特兰数。原因是 C_i 保证任意时刻向左走的次数不超过向右下走的次数，组合数在所有的步数中选择 $n-i$ 个向左下走。

复杂度 $O(Tn)$，因为模数不固定所以时限较大，为了卡可能不存在的多项式算法。