T1

测试点 1：送分，直接输出 1 即可。

测试点 2：送分，答案必然是 2 1。

测试点 3 ~ 4：稍微讨论一下也可以得到答案。

测试点 1 ~ 10：实际上，可能的出栈序列只有 $C(n)$种，其中 C 为卡特兰数。根据暴力算法实现的好坏，可以得到 0~50 分不等的分数。

测试点 1 ~ 17：考虑贪心，对于当前的栈顶元素 x，与所有未入栈的元素最大值 y。若 $x < y$，则继续入栈直到 y 入栈，然后弹出 y，否则弹出 x。由于朴素选取最大值的时间复杂度为 $O(n)$，因此算法时间复杂度为 $O(n^2)$。根据实现的好坏，可能得到不同的分数。

测试点 1 ~ 20：注意到测试点 1~17 的算法中，随着入栈元素越来越多，y 必然单调下降，因此只需要利用此单调性，即可把选取最大值的操作做到均摊 $O(1)$，这样总时间复杂度变为 $O(n)$，详细实现可以参考标程。

T2

测试点 1 ~ 2：答案显然是 0。

测试点 3 ~ 6：不妨设 $v_1 < v_2$，则第 2 只思考熊先跑完比赛，比赛时间 $T = \dfrac{LA}{v_2}$。扣 1 圈所需的时间为 $t_0 = \dfrac{A}{v_2 - v_1}$，因此套圈事件的发生时刻必然为 kt_0，其中 k 为正整数。所以总的套圈次数就为 k 的最大值。注意到 $k_{\max} t_0 \le T$，联立以上各式得：$k_{\max} = \left\lfloor \dfrac{L(v_2 - v_1)}{v_2} \right\rfloor$（与 A 无关），答案即为 $k_{\max}$。

测试点 1～10：对于一般情况，比赛时间$T=\dfrac{LA}{v_{\max}}$，其中$v_{\max}$是所有思考熊中速度最大的那个。将所有思考熊按照速度从小到大排序，考虑第 i 只与第 j 只($i<j$)思考熊之间发生的套圈次数a_{ij}，类似于测试点 2 的分析方法可得：$a_{ij}=\left\lfloor\dfrac{L(v_j-v_i)}{v_{\max}}\right\rfloor$。而我们要求的总套圈次数就是$\sum\limits_{i=1}^{n}\sum\limits_{j=i+1}^{n}a_{ij}$。这样做时间复杂度为 $O(n^2)$。

测试点 11～14：当 L 是$v_{\max}$的倍数时，$\dfrac{L(v_j-v_i)}{v_{\max}}$必然是整数，因此$a_{ij}$计算公式的下取整可以直接去掉，所以$a_{ij}=\dfrac{L(v_j-v_i)}{v_{\max}}=\dfrac{Lv_j}{v_{\max}}-\dfrac{Lv_i}{v_{\max}}$，进而我们可以得到：$\sum\limits_{i=1}^{n}\sum\limits_{j=i+1}^{n}a_{ij}=\sum\limits_{i=1}^{n}\sum\limits_{j=i+1}^{n}\left(\dfrac{Lv_j}{v_{\max}}-\dfrac{Lv_i}{v_{\max}}\right)=\sum\limits_{i=1}^{n}\left[(n-2i+1)\times\dfrac{Lv_j}{v_{\max}}\right]$。排序的时间复杂度为 $O(n\log n)$，计算答案的时间复杂度为 $O(n)$。

测试点 1～20：考虑更一般的情况。我们首先将a_{ij}的下取整打开，直接计算去掉下取整以后的答案$A_1=\sum\limits_{i=1}^{n}\left[(n-2i+1)\times\left\lfloor\dfrac{Lv_j}{v_{\max}}\right\rfloor\right]$，进而我们考虑这个值与正确答案之间相差多少。不妨设Lv_i对$v_{\max}$取模的结果为m_i。则从a_{ij}的计算公式中我们可以发现：

- 当$m_j\ge m_i$时，$a_{ij}=\left\lfloor\dfrac{L(v_j-v_i)}{v_{\max}}\right\rfloor=\left\lfloor\dfrac{Lv_j}{v_{\max}}\right\rfloor-\left\lfloor\dfrac{Lv_i}{v_{\max}}\right\rfloor$。
- 当$m_j<m_i$时：$a_{ij}=\left\lfloor\dfrac{L(v_j-v_i)}{v_{\max}}\right\rfloor=\left\lfloor\dfrac{Lv_j}{v_{\max}}\right\rfloor-\left\lfloor\dfrac{Lv_i}{v_{\max}}\right\rfloor-1$。

我们注意到$m_j\ge m_i$与$m_j<m_i$这两种关系将导致a_{ij}相差 1。实际上，我们只需要计算有多少对$m_j<m_i$便可知道需要减去多少个 1。注意到我们之前的条件 $i<j$，因此我们求的其实是 m 数组中逆序对的个数，记其值为 A_2，那么我们最终的答案就是A_1-A_2。排序时间复杂度为 $O(n\log n)$，计算 A_1 的时间复杂度为 $O(n)$，计算 m 数组的时间复杂度为 $O(n)$，计算 A_2 的时间复杂度为 $O(n\log n)$。因此这道题最终在 $O(n\log n)$的复杂度下得以解决。

T3

考虑预处理任意两点间的距离，这里使用 floyd 即可。接下来依次枚举 x 和 y，然后 $O(n)$计算答案。时间复杂度 $O(n^3)$，期望得分 30 分。

可以证明一旦选定了 x 和 y，则存在一种行走方案，使得聚集在 x 的区域彼此连通，聚集在 y 的区域也彼此连通。这个可以通过反证法证明，留给同学们自己思考。

这样，我们只需要把树分成两部分，然后在每部分选择一个聚集点，使得这部分的点到达聚集点的代价和最小。显然，聚集点选择树的重心。这样我们只需要枚举一条边，然后把整棵树断成两部分，每部分求出重心并计算答案。注意到求重心的复杂度为 $O(n)$，因此这个做法的时间复杂度为 $O(n^2)$，期望得分 50 ~ 60 分。

考虑更优的做法。

我们首先以 1 为根建树，记 $S(i)$表示以 i 为根的子树的人数总和，设 j 是 i 的一个儿子，可以计算出将聚集点从 i 变成 j，会使得总转移代价增加 $d = S(i) - 2S(j)$，则只有当 $d<0$ 时，才会使聚集点由 i 向 j 移动。注意到满足 $d<0$ 的 j 显然最多只有一个。这样我们预处理以 i 为根的子树的重心是哪个点，i 为根子树的答案（就是最小的 T），以及 i 的儿子中 $S(j)$最大的是那个点，然后考虑断开树中的一条边。此时，有一棵树是完整的，另一棵则是原来的树去掉这棵完整的树得到的。那么，这时候最大的 $S(j)$有可能会变小，进而被次大的 $S(j)$代替，于是我们需要记录一个点最大的 $S(j)$和次大的 $S(j)$。

注意到上面的做法与树的深度有关，设树高为 h，则时间复杂度为 $O(nh)$。由于数据的随机生成，所以树高的期望值为 $O(\log n)$，因此该算法的复杂度为 $O(n\log n)$。

有兴趣的同学可以思考该题目严格 $O(n\log n)$的做法，但这已经超出 NOIP 的范围。

另外数据设有梯度，鼓励各种奇怪的骗分算法。

T4

测试点 1～2。由于字符串长度不超过 2，因此不会出现括号，也不涉及到等价变换，将输入直接输出即可，期望得分 10 分。

测试点 3～4。由于字符串长度不超过 5，因此字符串中最多只出现一对括号，简单判断是否交换能得到更小的字典序，期望得分 20 分。

测试点 1 ～ 8。这部分测试点中字符串的长度不超过 20，首先解析输入的字符串，找到每个右括号匹配的左括号，然后直接爆搜就可以了，可以使用一个 map 来进行判重，直接将搜过的字符串扔进 map，直到没有新字符串产生，期望得分 40 分。

测试点 1～16。将输入字符串的括号进行匹配，找到每个右括号对应的左括号。记匹配后的字符串为 `S=aaa[A][B]`，其中 `A` 和 `B` 都为 `expr`，`B` 右侧的括号为整个字符串的结尾，`B` 左侧的括号是与最后的右括号匹配的左括号。定义过程 `solve(S)`表示寻找字符串 `S` 的最小字典序表示，首先递归调用 `solve(aaa[A])`与 `solve(B)`然后再比较 `aaa[A][B]`与 `B[aaa[A]]`的字典序，将 `S` 变为这两者中较小的一个。时间复杂度 $O(n^2)$，期望得分 80 分。

测试点 1～20。在 80 分做法的基础上，使用链表控制整个字符串，记录每个字符串的后继元素，这可以使得等价变换的时间复杂度变为 $O(1)$，期望得分 100 分。