Solution: Homework 1

1299. Closest pair

任意实现一个正确的最近点对算法即可。

注意到给出的样例 Python 代码复杂度是 log 方的，$O(n \log n)$ 实现里应该在分治时对 y 进行归并排序或者其他小技巧。

如果你对着 Python 代码一行一行确认无误却发现超时时：

- Python 的 list 是引用传递的，所以，请注意不要在 C++ 程序里传递 `vector <point>`，应该是 `point[]` 或者 `vector <point> &`。
- 如果你手写的快速排序，那么有可能在各种奇怪的边界写挂：没有办法处理排好序的数组（比如选择第一个或者最后一个作为 pivot），没有办法处理全部一样的数组（比如 partition 的时候没有处理好导致返回数组头 / 尾），这边建议是写 merge sort 或者写比较函数调用 `std::sort`。
- 申请数组的时候，注意一下总共申请的大小：分治时申请一个 new point [r] 可能会导致总大小达到 $O(n^2)$ 导致超时。在写算法题的时候，可以大胆使用全局 / static 数组，这在处理这种问题的时候并不是一种不好的习惯。特别是分治需要临时数组时，你可以发现一个全局或者 static 数组能够每部分被完全利用，比每次申请内存听起来美好不少。

最开始的数据让下面这个做法过了：

- 将每个点按 x 排序，然后每个点向前找 x 上相邻的点，一旦 x 轴上的距离大于最短距离，就 break

然后显然一排在一行 / 一列上的点会让这个做法复杂度达到 $O(n^2)$。

同时，本题调用了 Probablistic Killer 中的曼哈顿网格图构造来造了卡掉 500 层前后扫描的数据。

当然，有一位同学在我连续两次加强数据 hack 掉他的程序后，坚定不移地继续编了一个乱搞水过去了。尽管还是可以卡掉，但是继续更新数据的话评测就要两分钟了，所以就放过去了。我真的怀疑有些人闲的程度啊.jpg

1300. k-th Smallest Number

正确的随机仍然是很有必要的。当然，由于输入也是随机生成的，你可以选择除了前 10^5 个数中的任意一个作为 pivot 都是 ok 的。

不过，如果你的算法不能通过前 m 个数为 $1 \cdots m$ 的话，那么会被卡掉——比如每次选第一个数或者最后一个数肯定是不行的。

选中点其实也不好，因为并不是随机的，但是放过去了。

Median of medians 常数蛮大的，从实际角度出发并不实用。 提交里的 quick select 典型消耗时间不超过 300 ms，但是 median of medians 跑得最快的也要 800 ms。

`std::nth_element` 是可以使用的，因此你可以一行通过这个题。

1301. Bubbling bubbles

冒泡排序每次交换都会使逆序对恰好减少 1，因此即求逆序对数量，不过贡献记在每个元素上。

一个典型的错误做法是：

- $\text{solve}(l, m); \text{solve}(m+1, r);$

- 数 $i \in [l, m], j \in [m+1, r]$ 里的逆序对。

稍加分析可以发现是 $O(n^2)$ 的。

1302. Optimal sort

将上一个题的归并排序粘过来就过了，因为

$$T(n) = 2T(n/2) + n - 1,$$

$n-1$ 为比较次数，类似于 $[1,3,5,7],[2,4,6,8]$ 的合并可以达到这个上界。

同时，类似于二分插入排序的做法亦是正确的，在把所有 $\log_2(i)$ 放缩为 $\lceil \log_2(i) \rceil$ 时后式子也是紧的。由于这个上取整在二分时总存在能取到的情况，因此二分插入排序最坏仍然会达到题面中的界。

注意：`std::stable_sort` 是 $O(n \log^2 n)$ 的，因为标准库实现了一个原地 (in-place) 排序，合并是 $O(n \log n)$ 的。因此，你不能一行通过这个题。

~~1303. Subset sum~~ [Deleted]

1304. Probabilistic killer

本题如题面所说，是 A 题的副产物。

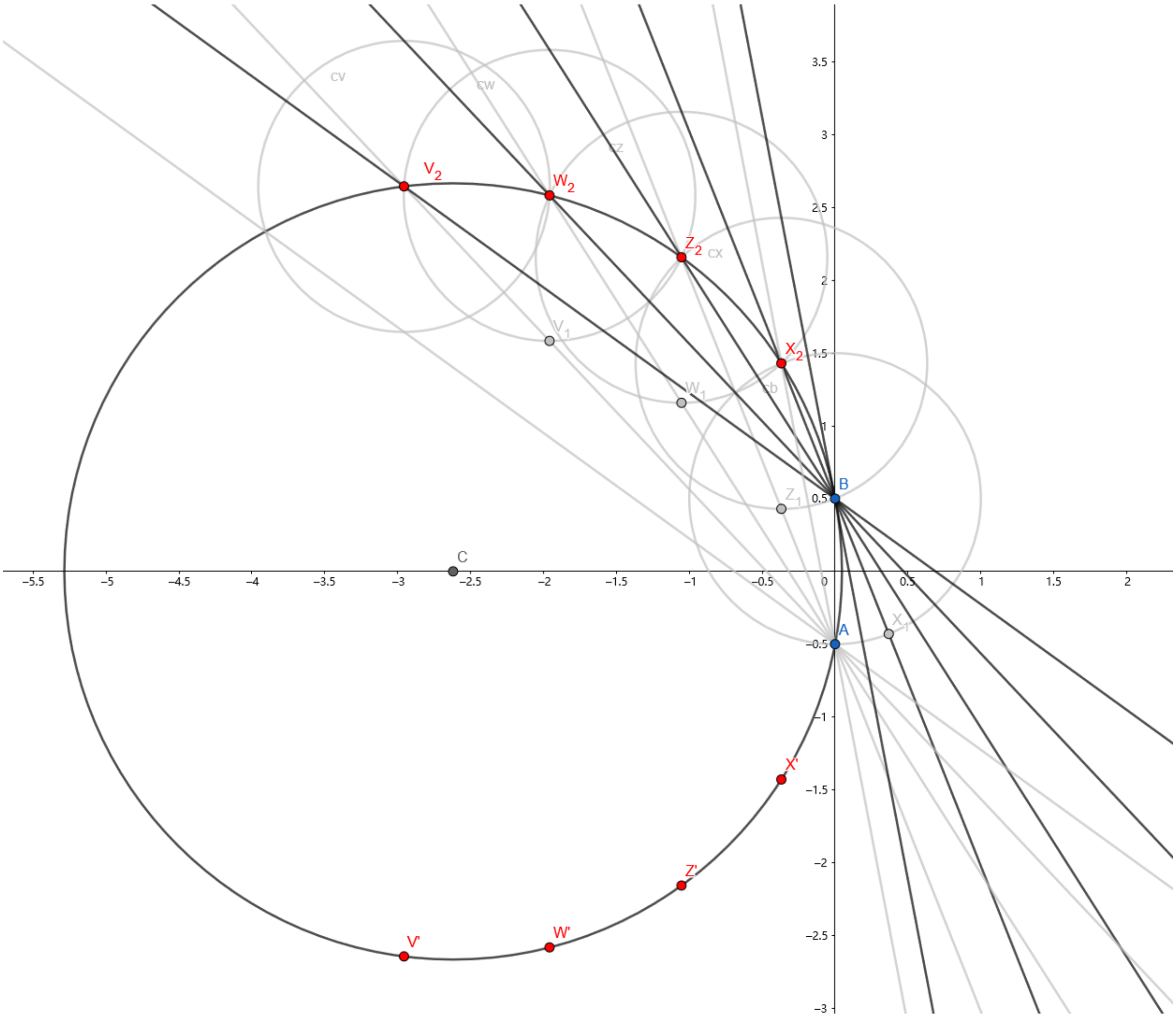
题面的算法可以描述为：随机选择几个方向，将所有点在方向的法向量上做投影，然后查询排序后相邻点的距离。

听起来非常正确，并且实际上也难以卡掉。在主流 OJ 上的许多最近点对问题可以用这个方法水过去。

但这个做法是不对的。

我们考虑钦定最近点为 $A = (0, -0.5)$ 与 $B = (0, 0.5)$，此时最近点对距离为 1。

结论：对于任意目标概率 $\varepsilon > 0$，存在一个构造方法，使得只需要使用至多 $1/\varepsilon$ 个实数点，便可将算法一次成功概率降低至 ε。这种构造方法的所有点分布在 A, B 以某个 y 轴上的圆心 C 形成的圆上，并且相邻两个点的圆心角恰为 $\angle ACB = 2\pi/\varepsilon$。下图为 $\varepsilon = 0.06$ 时，y 正负区域的前四个点。图中剩余部分是原始构造所使用的辅助线。



下面我们对思路进行解释。

假设对于某个方向 θ，在排序后最近点不相邻，那么也就是过 A, B 做 θ 角度的两条直线之间夹了一个坏点，算法就挂了。

对于任意一个坏点，他能覆盖的实际上是一个角度区间，我们希望用尽量少的点覆盖尽可能大的角度区间。

同时，除了最近点对外，所有点周围 1 的距离内不能有任何其他点。

一个非常直观的想法是：我们围绕这两个点作一个大圆，上面均匀撒一些距离略大于 1 的点，以在每个角度都能干扰到算法。

但是这么做是错的，因为：

- 对于每个方向上的坏点，越接近原点，那么覆盖的区间越大。大圆太远的话，覆盖区间很小，效果有限；
- 如果 θ 贴近 x 轴，那么算法很容易失败；如果 θ 贴近 y 轴，那么算法很难失败。如果恰好有点在 y 轴附近，那么算法将会直接失败。因此，大圆上均匀分布并不会使得坏点很均匀。

因此我们尝试从覆盖区间入手考虑。由于贴近 y 轴的更坏，我们从 y 轴出发，依次构造出所有点。

以上图为例，我们放弃掉 y 轴周围 $\theta = \varepsilon\pi$ 的角度，过 A 作与 y 为 θ 的直线，交 B 为圆心 1 为半径的圆上为 X_2。此时，X_2 即为第一个确定的点，其相对 y 正半轴的角度区间为 $[\angle yAX_2 = \theta, \angle yBX_2]$。

考虑下一个点：其角度区间起点需要是 $\angle yBX_2$，那么我们过 A 作与 y 为 $\angle yBX_2$ 的直线，交 X_2 为圆心 1 为半径的圆上为 Z_2。此时，Z_2 即为第一个确定的点，其相对 y 正半轴的角度区间为 $[\angle yAZ_2 = \angle yBX_2, \angle yBZ_2]$。

可以发现 X_2 和 Z_2 的构造方法是完全相同的，因此可以一直做下去。

这个方法有以下美妙的性质：

1. 上文提到的所有角度区间大小均等于 θ，可用圆、菱形、平行线证得；
2. 上文提到的所有点共圆，可用四点共圆证明。

因此，这样的构造必然会在得到 $1/2\varepsilon$ 个点后停止，对称构造负半轴，则一共 $1/\varepsilon$ 个点。

尽管我很想给这个圆上的构造一个几何解释，但是我的几何水平有限，很难找到一个合适的几何意义。如果有人能找到更好的解释并告诉笔者，我将感激不尽。

因此，本题的参考解法就是随便在哪里画一个圆就好，最好需要微调一下让最近点对唯一。

其实可以做到使用的点数更少：注意到上图中远离 y 轴的部分都可以移动到靠近 y 轴的部分，来让覆盖的角度区间尽量大。当然，这样的话解法就蛮丑了，因此范围就设为了圆能过的范围。

由于本题需要整数点，将格子放大（例如 $A=(0,-5000), B=(0,5000)$）然后取整即可。为了防止取整误差导致答案变化，可以在输出时将 A,B 稍微往原点靠近一点。不过由于取整误差的存在，需要多放一些点来规避误差（理论上需要 500 个点即可，但由于取整误差，其实在 800 个点左右才满足题目要求）。

本题乱搞做法效果应该挺差的，因为题目中的 sample 方式相当于要求了不能有连续超过 $1/T$ 的角度区间坏掉，这通常意味着你能够通过最后几个点的话，你的构造已经足够强了。笔者构造了一个曼哈顿网格覆盖，用角度区间贪心去重，调了很久，才通过了 60 分的数据。

以下是同学们提交的题解，其中：

- 李志腾同学的做法是与题解不同的：他的做法是从与题解相反的方向开始的构造，并且最终只取落在两条平行于最近点对的直线上的点。由于构造非常精妙，在边角处没有浪费，正确率达到了 $\frac{\tan^{-1}(4/1250)}{\pi/2} \approx 1/490$。
- 王煌基同学用很自然的思路命中了正解，也做了一些精度上了分析。
- 宋雅昆同学展示了这道题思考的思路，与参考解法的方向非常类似。
- 还有一位同学提交了他的大作，非常精彩。

李志腾

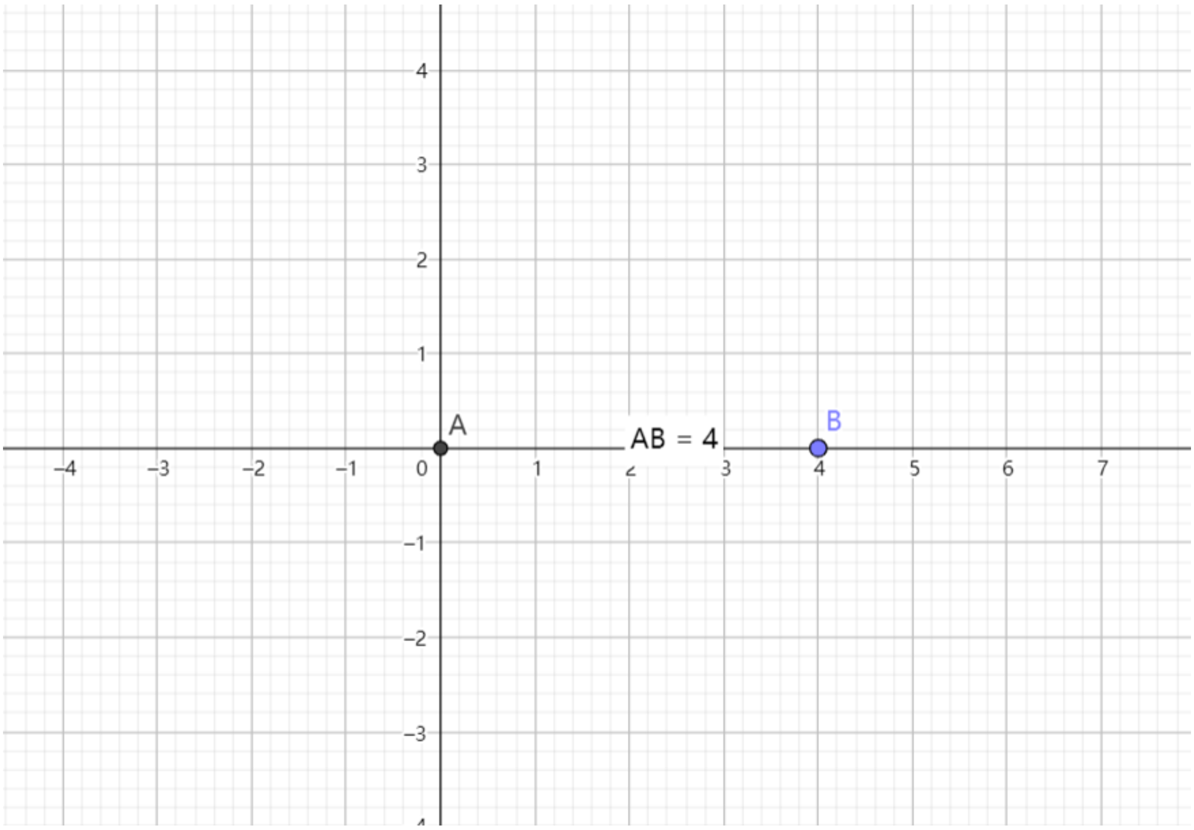
问题分析

closest pair 的随机化算法将所有点绕原点逆时针旋转特定角度后，按照 x 轴投影的坐标重新排序，以相邻两点间的欧式距离更新最短距离。

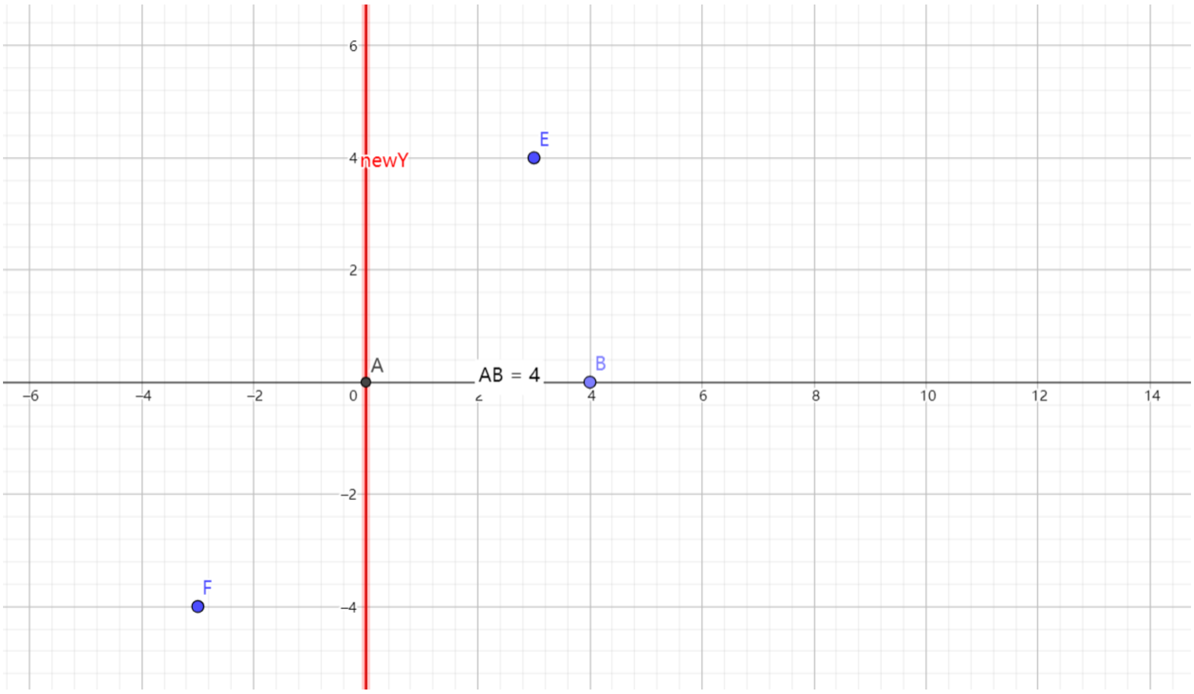
将所有点绕原点逆时针旋转可以转换为将坐标轴绕原点顺时针旋转；由于该算法仅按横坐标（也即某点离 y 轴的距离）进行排序，我们可以只考虑 y 轴绕原点的顺时针旋转。

算法

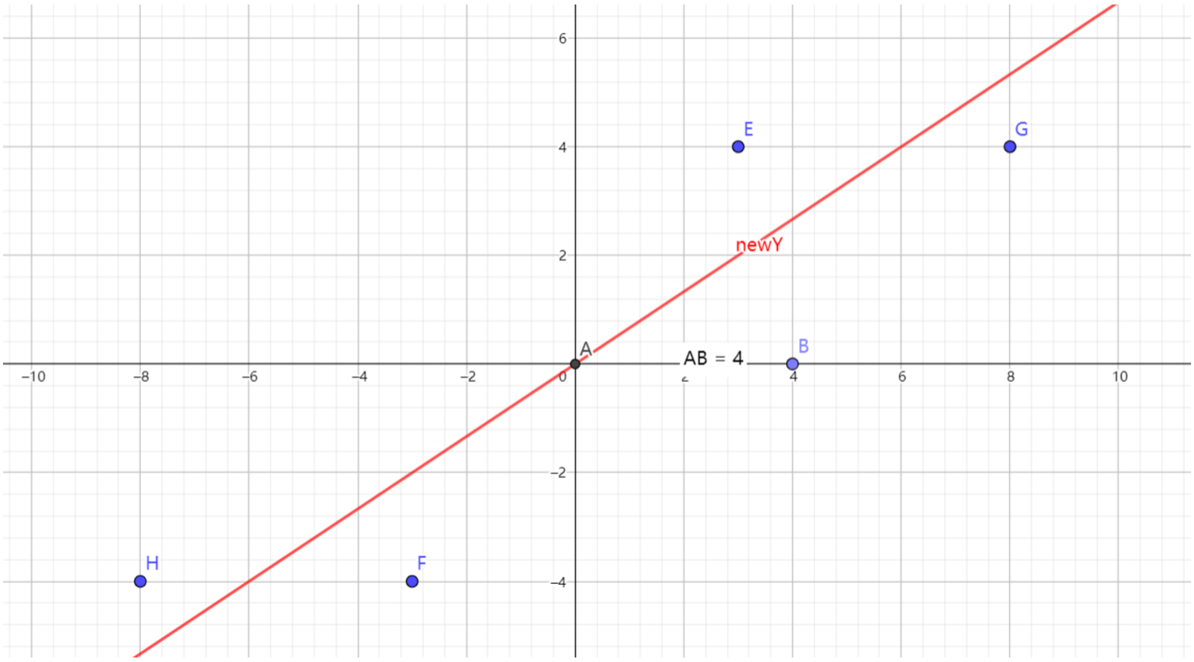
首先选择两个点作为 closest pair（如 $A(0,0)$ 和 $B(4,0)$），然后开始旋转 y 轴，接下来只要每次构造一对到 y 轴的距离小于 B 点到 y 轴距离的点即可，同时还要保证构造的这一组坐标点与已存在的点之间的距离大于 AB 之间的距离。



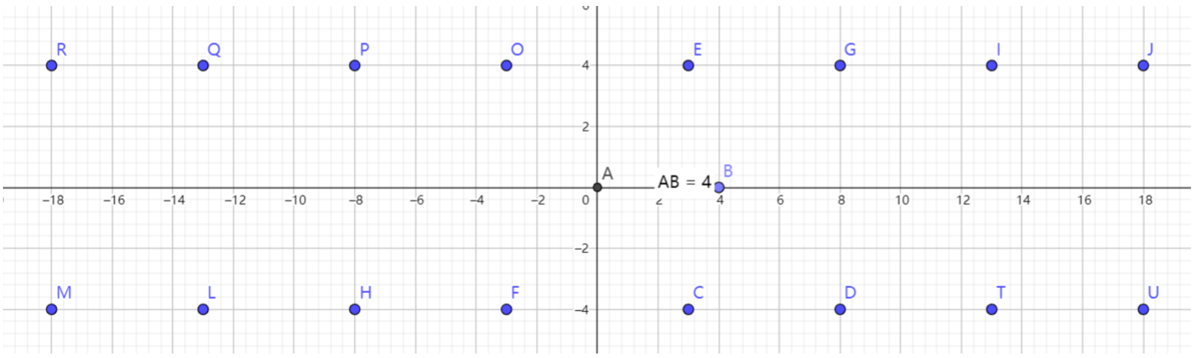
初始状态 y 轴顺时针旋转 0°, 选取 $E(3,4)$ 和 $F(-3,-4)$ 一组对称点，使其为当前到 $newY$ 轴距离最近的两点。



接下来顺时针旋转 $newY$ 轴，当接近 $\blacktriangle AEB$ 的角平分线时，继续构造对称点对 $G(8,4)$ 和 $H(-8,-4)$， 使其为当前到 $newY$ 轴距离最近的两点。重复以上步骤，每次添加一组对称点对，使 B 点在 $newY$ 轴旋转过程中始终不是到 $newY$ 轴最近的两点之一。



不难发现，只要每次都将第一象限的点横坐标+5，第三象限的点横坐标-5，就能达到上述目的。因此最后构造的点对如下：

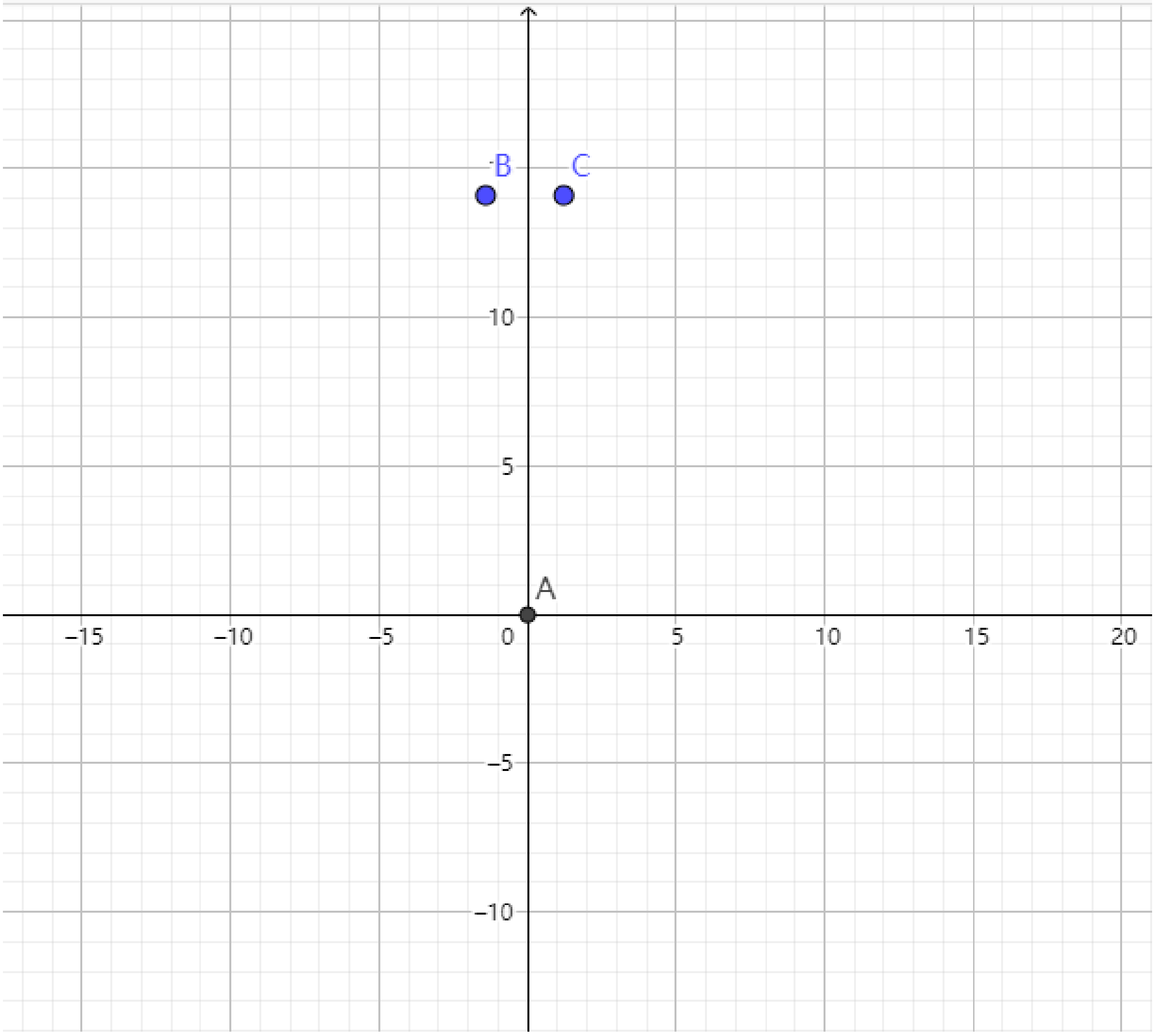


测试结果

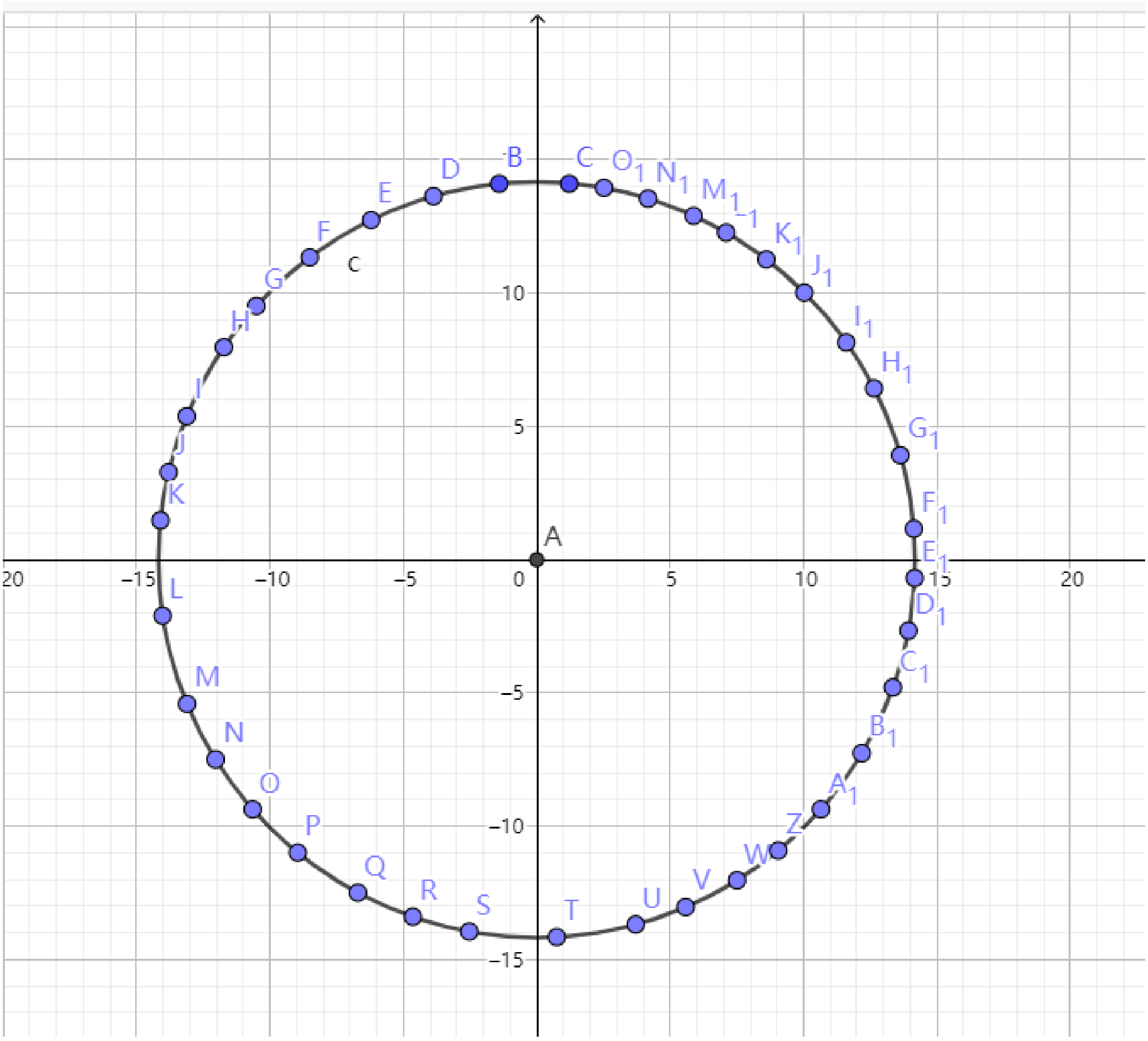
限于题目的要求，我们最多只能构造1000个点，因此当 $newY$ 轴旋转到原 x 轴附近时， B 点仍将是离 A 点最近的点，但题目只要求在 20 次执行中让随机算法错误两次即可，因此该构造方法仍能通过。

王煌基

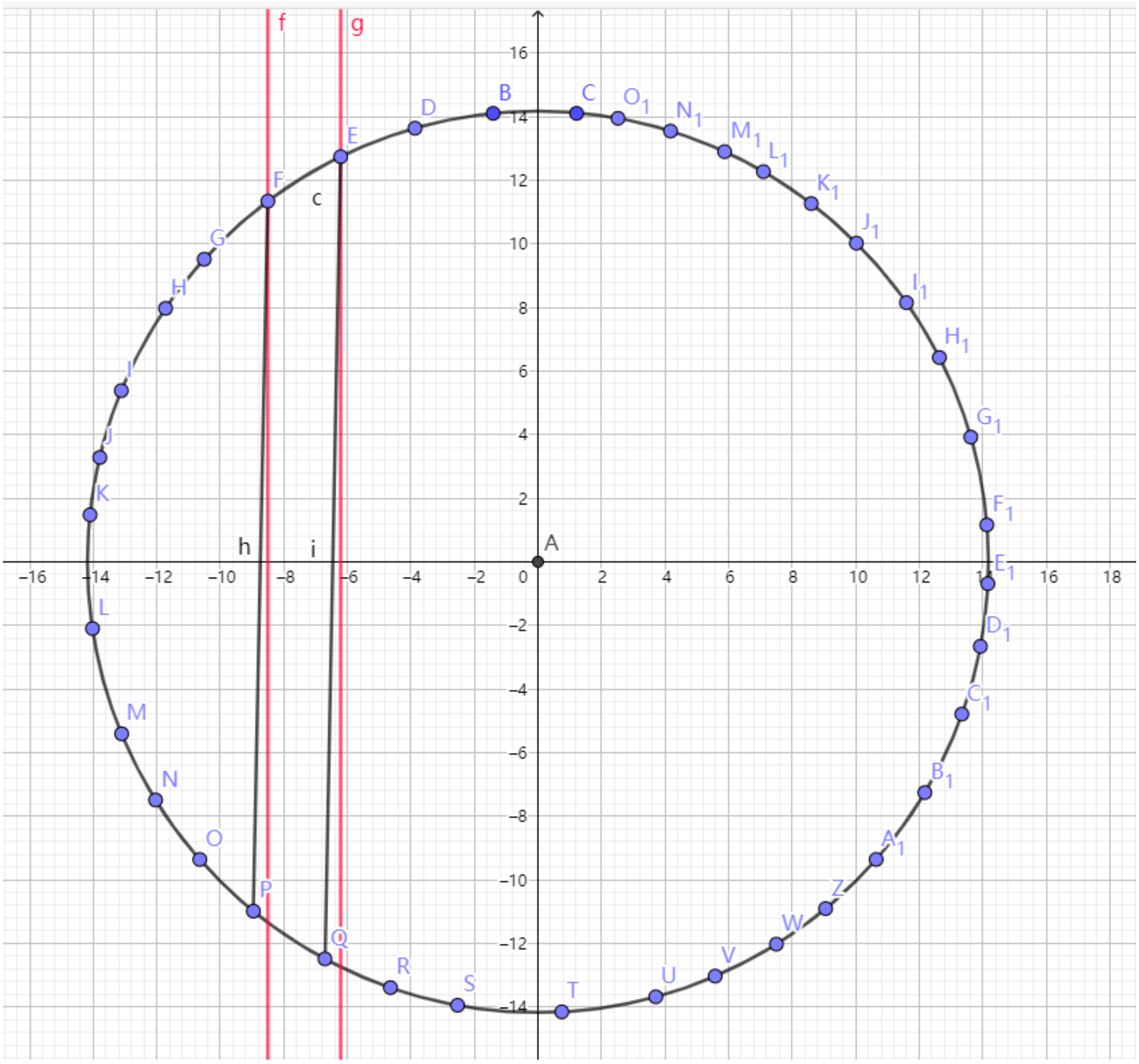
由于这道题与随机旋转有关系，因此首先想到的就是构造一些旋转过程中的特例。如下图所示，我最初的想法是在原点构造一个点，然后在距离原点无穷远的地方构造两个离y轴相当近的两对称点B,C，这样一来在最初的情况下实际上直接计算肯定是BA和AC而不会计算BC，当然，稍微旋转一下就寄了。



后面我把这个情况扩展到n个点，然后以圆的形式就能确保在很大概率的旋转下是保持不变的，这样一来，在旋转的过程中应该很多量是不会变的，包括它们x轴上的相对位置可能不容易改变。



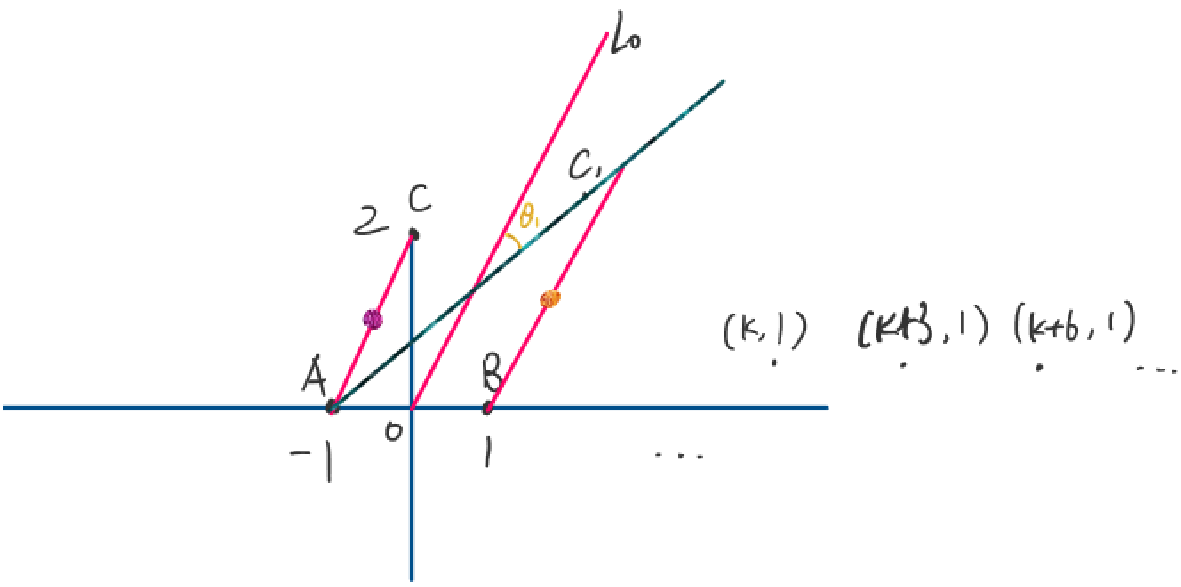
然后，基于long double（长实型）与long long（长整型）之间转换过程中可能出现的精度差异，在某种可能的情况下（离散点构造出的圆形足够圆的话），我应该能够构造出来一种圆，它会使得每一对点之间会出现即使相邻，但是它们永远不会被计算到一起。具体情况应该长成这样（其中，红色为垂线而黑色为链接的线段）：



因此，按照这种方法构造出来的点将只会计算到如EQ，FP以及最边缘的KL这种上下对称的点，不会涉及到左右相邻的点对，这样的方法就可以成为一个随机算法的BUG。

我最初以为我能AC是因为long double和整数转换直接有丢失，导致结果经常有丢失（如：321215456和321215768），然而事实应该是我上图中所说的情况。后面我通过多次提交点的构成数目就可以确定了n的最优情况。

宋雅昆



假设我们希望随机算法不能发现A,B是最近点对。

为了方便说明, 首先假设 C(0,2) 是已经生成的用于hack的点，然后，只说明第一象限的情况。

分别过O、B做AC的平行线，令y轴从OC开始旋转，旋转到l0时，A、C离旋转后的y轴有相同距离，一旦继续旋转，就会使得A点离旋转后的y轴近于C点，此时AB点会成为x轴上相邻的点。为了hack掉这样的情况，我们可以在la和lc这两条线之间补充一个点C1，只要这个点离已生成的所有点的距离严格大于2【*】，就可以了；当然，这个点不要在AC和l0之间：这样会生产出一个废点。然后对C1继续上述步骤，就能生成所需的点……

但是这种方法，OCn很快会收敛到第一象限角平分线，所以是不行的。此外，点C1实际“控制”的夹角是θ1，这里“控制”指的是，旋转到某些角度时，因为加入了这个点，这些角度能hack掉，而不加这个点则hack不掉。

我们自然希望每次都取尽量大的θ，但是这很难。而且，我们应该尤其关注y轴旋转到接近原先x轴的情况：为了hack掉这样的情况，我们需要仰角很小的点，但是这样的点能“控制”的角度非常非常小。所以，我们应该额外“关照”这样的情况：尽量使他们能多控制一些区域：即把（3，1），（6，1），（9，1），...，（？，1）无条件地分配为初始点。这里？的选择，完全取决于我最终想要放弃哪个角度范围的情况（比如，如果我最远的点选择为（500，1），那么我基本就放弃了y轴旋转到y=1/501以下的情况：该如何放弃取决于T值；你不得不做出取舍）。

除此之外，我还允许在生成点时，适当随机产生一个微小的不被控制的区域。用上图举例，当我用A、B、C三个点去生成C1时，我允许C1出现在lb的右边（也即下边）一点点（用角度衡量的一点点），这个使得生成算法可以避开收敛。

最终方案是上述思想的结合。这其中有很多可以调整的地方，比如可以调整初始“无条件分配”的点，使其变为交错的2排；可以调整宽恕角（这个名字有点搞笑）的大小。生成第一象限的角以后，将其对称到第二象限即可。第三和第四象限可以一个点都不放，但更好的方法是，放一些不满足【*】，但是很优秀的点（如果可以放）。这里的策略也有很多种。

？？？

Bonus Answer：
I have a great idea for this problem, but the space is so small that I can not write here.
