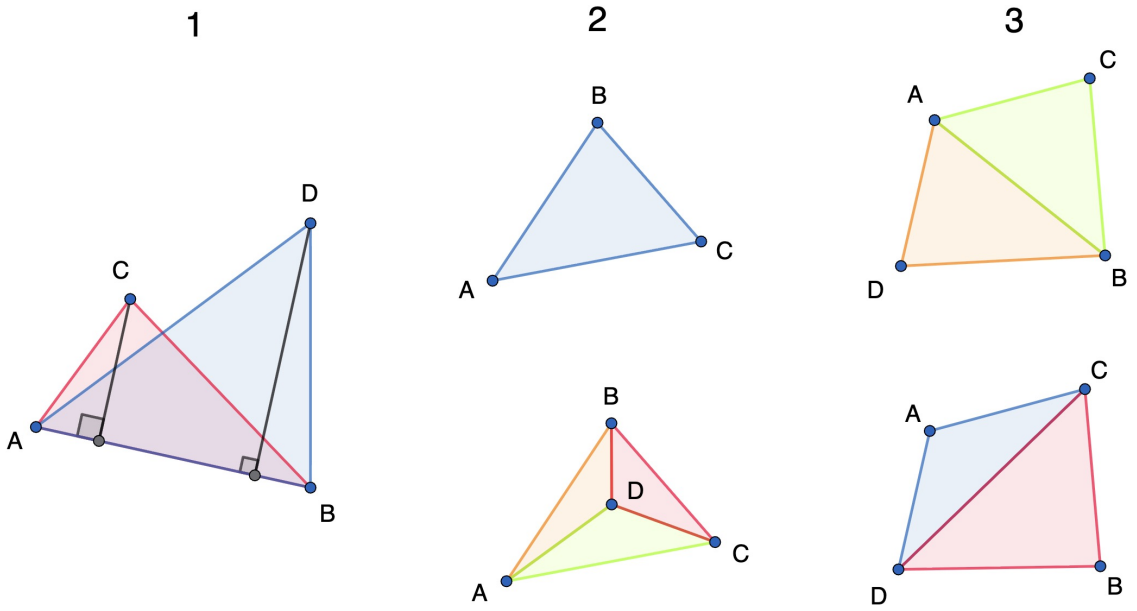
Cutting the Cake

Author and problem developer: Ivan Safonov

Note that the problem asks to construct a partition of the convex hull of the set of points into parts. For the groups without maximization, it is enough to find the convex hull; for the groups with maximization, one needs to find a triangulation of the set of points.

Note that with our query we can perform the following operations:

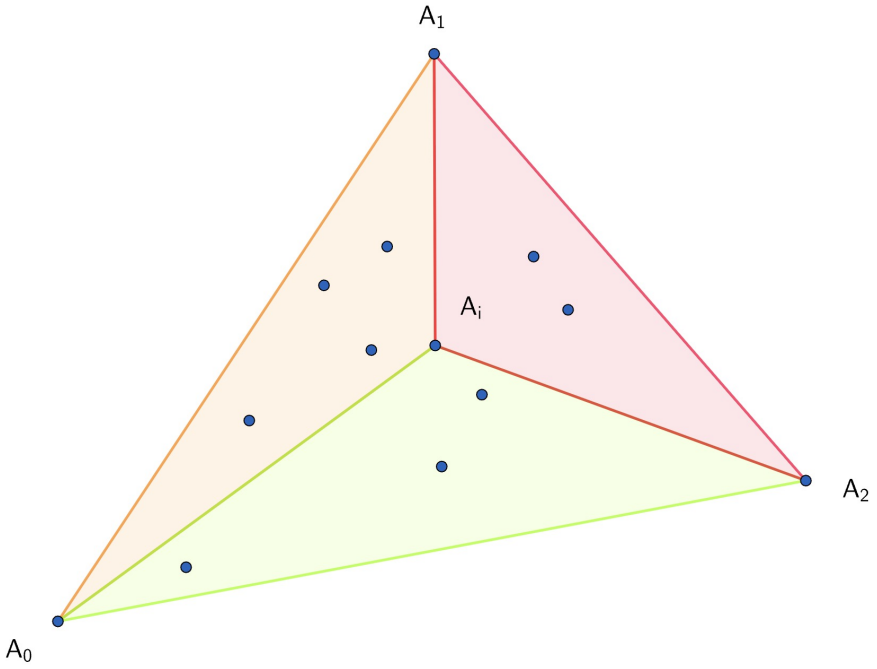
1. Given points A, B, C, D, determine which of the distances $\text{dist}(C, AB)$, $\text{dist}(D, AB)$ is larger (the distance to the line). To do this, it is enough to compare $S(ABC)$ and $S(ABD)$.
2. Given points A, B, C, D, determine whether point D lies inside triangle ABC. To do this, it is enough to check whether $S(ABC)$ equals $S(ABD) + S(ACD) + S(BCD)$.
3. Given points A, B, C, D, determine whether AB and CD intersect. To do this, it is enough to check whether $S(ABC) + S(ABD)$ equals $S(ACD) + S(BCD)$. It is exactly in this point that we use the fact that $ABCD$ is not a trapezoid, because for trapezoids equality may hold without AB and CD intersecting.



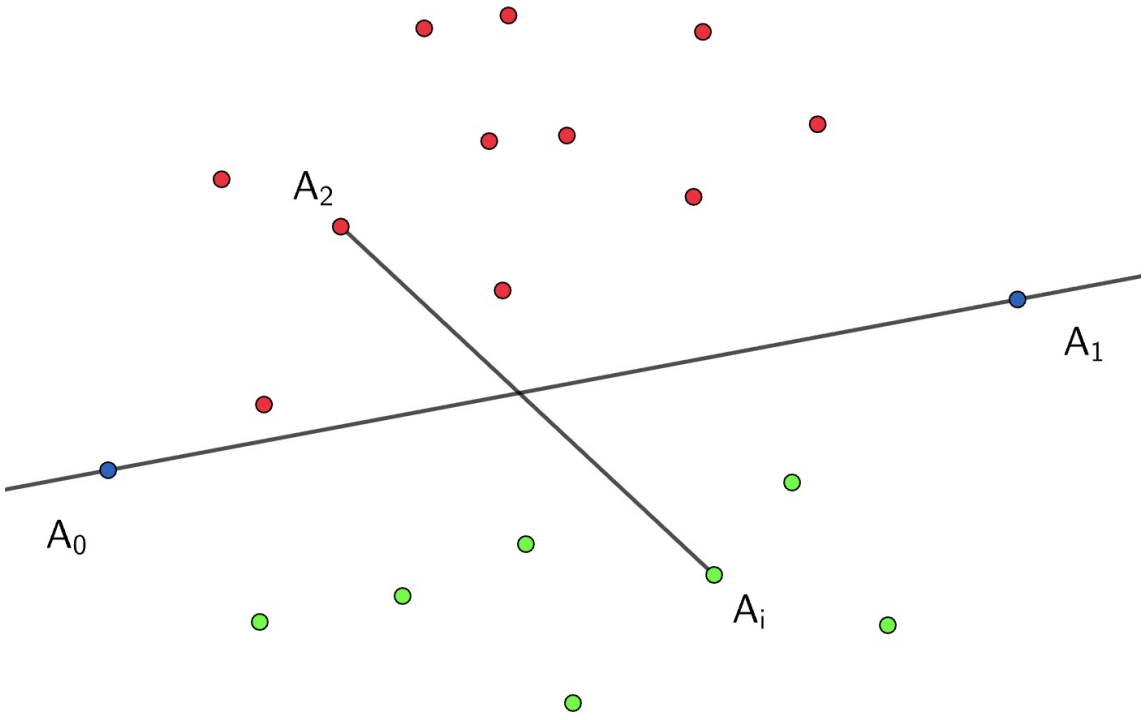
To solve groups 1 and 2 ($n = 4$), one can simply check whether each point lies inside the triangle formed by the other points. If so, the triangulation contains 3 triangles. Otherwise, the points form a convex quadrilateral. Its partition into 2 triangles can be found, for example, by checking which two segments intersect (they will be the diagonals of the quadrilateral).

Note the following idea: one can find a point lying on the convex hull in $n-2$ queries. To do this, take any two points, for example, A_0 and A_1, and among all the remaining points find the triangle with maximum area $S(A_0A_1A_i)$. Then A_i is definitely on the convex hull, because the distance $\text{dist}(A_i, A_0A_1)$ is maximal.

Thus, one can find three points on the convex hull in $\leq 3n$ queries. In groups 3 and 4, this will be the entire convex hull. To find a triangulation in this triangle in group 4, one can, for example, choose a random point inside it; it splits the triangle into 3 smaller triangles. Then, using queries of type 2, for each of the remaining points one can determine which of them it belongs to, and solve the problem recursively. Since the points inside the triangle are random, this will require $O(n \log n)$ queries.



To solve the next groups, first find two points A_0, A_1 (WLOG their indices are such) that lie on the convex hull. This will require $\leq 2n$ queries. Next, all remaining points can be split into two sets of points lying in different half-planes with respect to the line A_0A_1. To do this, one can take a third point A_2 and check each remaining point A_i by testing whether segments A_0A_1 and A_2A_i intersect. This will require another $\leq n$ queries. Then we will solve the problem for each of the half-planes independently. Therefore, now we assume that all points lie on one side of the line A_0A_1.



Sort the points by their distance to the line A_0A_1. This will require $\leq n \log_2 n$ queries.

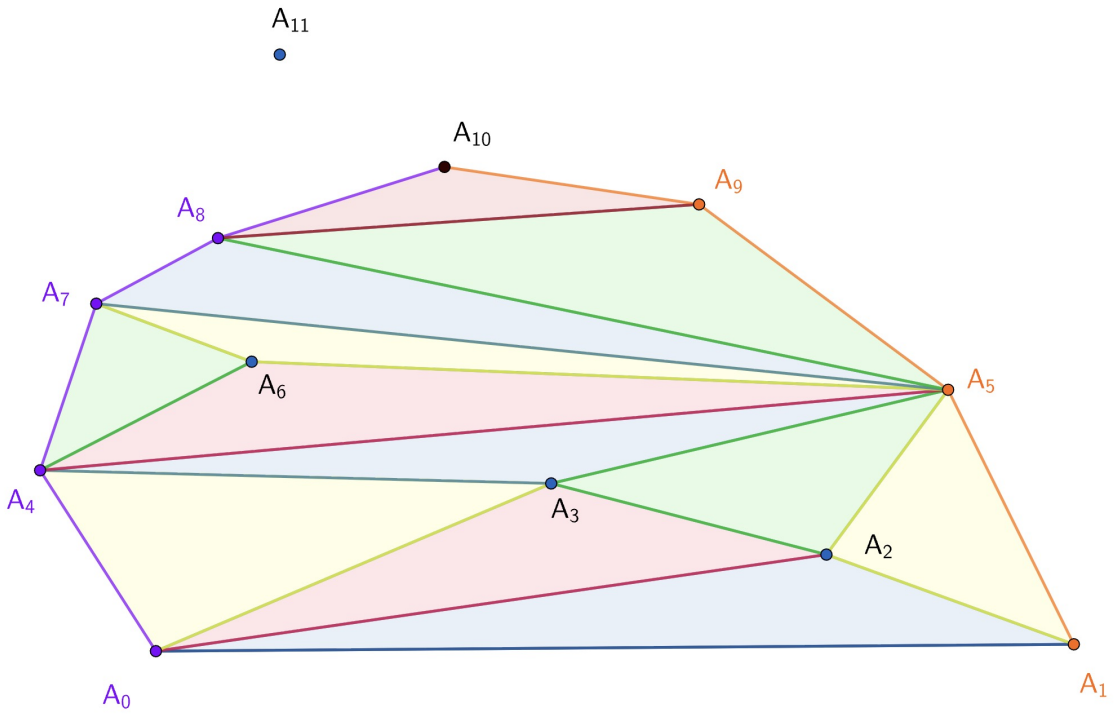
In group 5, one can take the farthest point A_i and again split all the remaining points into those lying on different sides of the line A_0A_i in $\leq n$ queries. Note that now the points in each part are monotone, and

we already know their order by height. Thus, the answer will be found in $\leq n \log_2 n + 4n$ queries, which fits comfortably.

Let us discuss the full solution. After sorting by height, apply an algorithm similar to Graham's scan, but adapted for points sorted by distance from a line rather than by angle. It turns out that during this algorithm we can also construct the triangulation.

Let the points be sorted by increasing distance from A_0A_1 in the order $A_2, A_3, \ldots, A_{n-1}$. After processing the prefix of this order $A_2, A_3, \ldots, A_{i-1}$, we will maintain the following state:

- We store the convex hull of the points $A_0, A_1, A_2, \ldots, A_{i-1}$ as the union of two chains, call them the left and the right one. The left one consists of points $[A_0, \ldots, A_{i-1}]$, the right one consists of points $[A_1, \ldots, A_{i-1}]$.
- We build a triangulation of this convex hull.

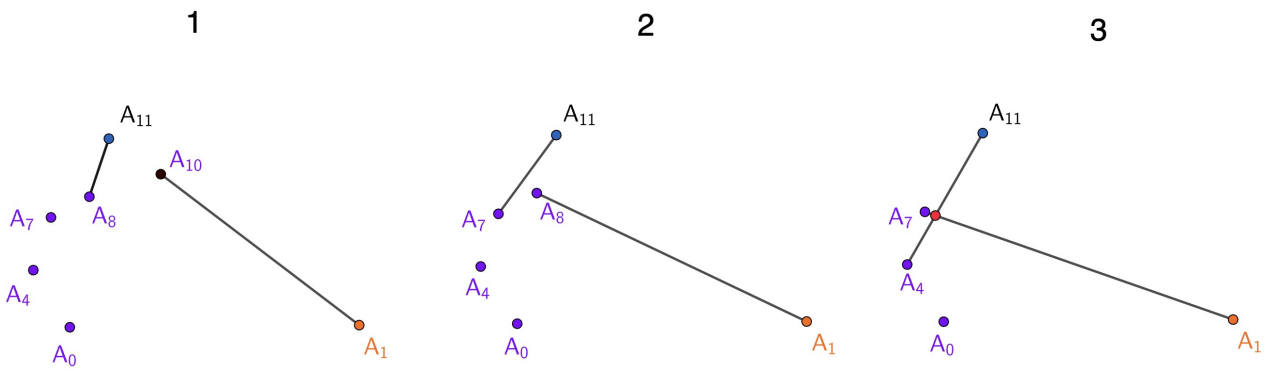


In this example, $i = 11$. The left chain consists of points $[A_0, A_4, A_7, A_8, A_{10}]$, the right chain consists of points $[A_1, A_5, A_9, A_{10}]$. We are adding point A_{11}.

Thus, in the end we will find the answer. For initialization (the only point A_2), build the left chain as $[A_0, A_2]$, the right one as $[A_1, A_2]$. The triangulation will contain a single triangle $A_0A_1A_2$.

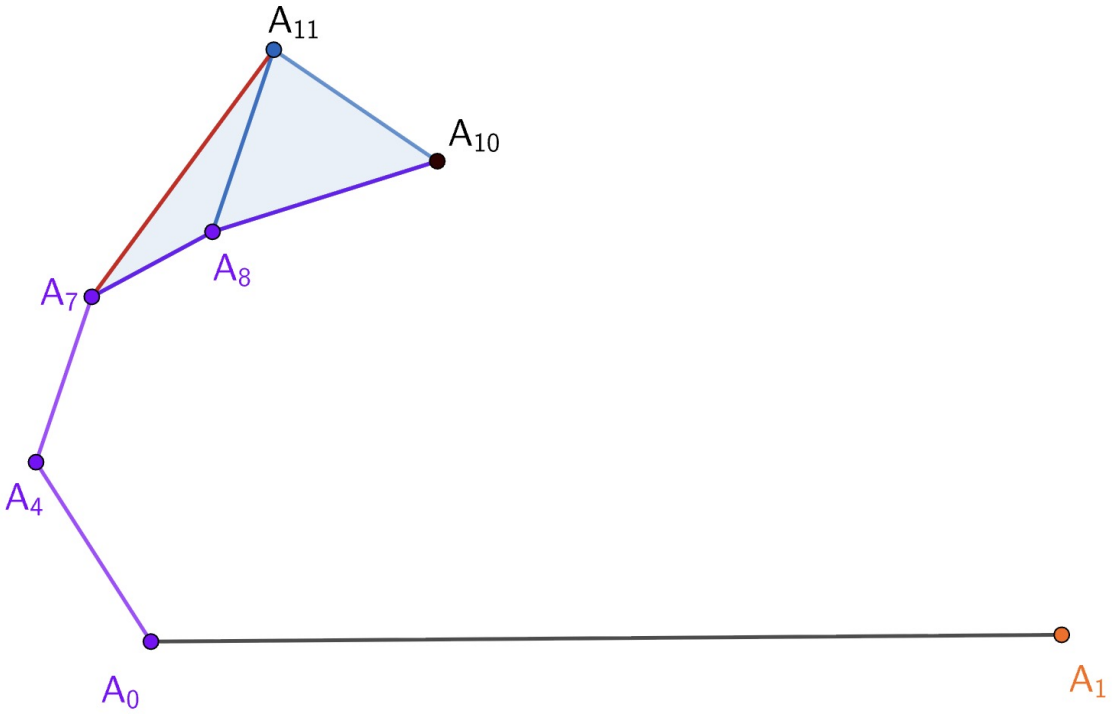
Next, let us learn how to add a new point A_i to this construction. Note that we need to add this point independently to the left and right chains.

Suppose the left chain is $[A_{l_0}, A_{l_1}, \ldots, A_{l_k}]$, where $l_0 = 0$, $l_k = i-1$. Note that now we need, as in the usual Graham scan, to remove some suffix of the chain and append A_i to the end. To remove the needed suffix, we should take the last two points A_{l_j}, $A_{l_{j+1}}$ and remove the last point $A_{l_{j+1}}$ if $\overrightarrow{A_{l_j}A_{l_{j+1}}} \times \overrightarrow{A_{l_{j+1}}A_i} > 0$ (cross product). Note that this is equivalent to saying that segments $A_{l_j}A_i$ and $A_{l_{j+1}}A_1$ do not intersect, which we know how to check using a type 3 query. Then let us remove the needed suffix of the chain using such checks.



When adding A_{11} to the left chain, we will do the following steps: do A_8A_{11} and $A_{10}A_1$ intersect? no, so remove A_{10}; do A_7A_{11} and A_8A_1 intersect? no, so remove A_8; do A_4A_{11} and A_7A_1 intersect? yes, so stop and append A_{11} to the end.

Moreover, whenever we remove the last point $A_{l_{j+1}}$ from the chain, let us add the triangle $A_{l_j}A_{l_{j+1}}A_i$ to the triangulation. Then these triangles will exactly cover the region formed between the tangent from point A_i and the old chain.



When updating the left chain, we add triangles $A_8A_{10}A_{11}$ and $A_7A_8A_{11}$ to the triangulation.

The same algorithm should be repeated for the right chain, but using point A_0 in the checks instead of A_1.

Thus, we will construct the convex hull and its triangulation. Let us note that this algorithm will require $\leq 2n$ queries for one chain, because each point can be added to and removed from the chain at most once. Therefore, this part of the solution will take $\leq 4n$ queries.

Thus, the entire solution requires $\leq n \log_2 n + 7n$ queries. If we estimate the actual sorting constant a bit more carefully when using merge sort, we can see that such a solution fits within $20n$ queries for $n \leq 10^4$. Note that the sorting estimate can be reduced a little more if, for sorting, we use an algorithm that spends $\leq \sum_{i=1}^{n} \lceil \log_2 i \rceil$ queries, by inserting a new index into the sorted order n times using binary search.