Code Jam 2018 - Round 3

Analysis: Raise the Roof

Test set 1

In Test set 1, the number of columns is small enough to allow a brute force solution. The problem guarantees that there is at least one correct order, so we can try all possible orders (there are at most 10! = 3628800) until we find a correct one. To check a given order, we check every prefix of length at least 3, build the roof implied by the last 3 columns of the prefix, and then check that all other columns are below it. Since the statement guarantees no 3 points are collinear, any subset of 3 points determines a single plane. There are several ways to check if a roof is valid; here is one that involves only integer arithmetic:

Let p_1, p_2, ..., p_k be the k points in the prefix. Let q be the plane that contains p_{k-2}, p_{k-1} and p_k. For each i between 1 and k - 3, inclusive, we need to check whether p_i is above q. Notice that we can subtract p_k from all points and the answer to the question doesn't change. Thus, we further assume p_k = (0, 0, 0). Let v be a normal to the plane q. Project p_i onto v via dot product and check whether v and the projection are on the same side of q. If they are, then p_i is above q, otherwise, it is not.

Since checking a prefix takes O(**N**) time, and there are O(**N**) prefixes to check for each of the **N**! orders, this yields an O(**N**! × $\mathbf{N}^2$) algorithm that may or may not be fast enough depending on your language and implementation. We can speed it up in several ways:

- Generate permutations by adding columns one at a time, and check each prefix as soon as it is generated. This reduces the complexity to O(**N**! × **N**) because each prefix is not re-checked for each permutation that starts with it. Additionally, when we find a prefix that does not work, we do not waste time checking other permutations beginning with that prefix, which can further reduce the search space in practice.
- Try a dynamic-programming-on-subsets approach to change the complexity to something like O($2^{\mathbf{N}}$ × $\mathbf{N}^4$), but whether or not that makes the solution faster in practice depends heavily on implementation details.

Test set 2

For Test set 2, any solution that is exponential or factorial in **N** is doomed from the start, so we need a different strategy. One possibility is to approach the problem in reverse: start with all columns, find a roof that works, remove a column, and repeat.

To formalize this a bit, we start by fixing the last two points q and p from the set of column tips. to be the last two columns, in that order. We discuss how to do this later. Then, we define s as the set of all column tips except p and q, and repeat the following until s is empty:

1. Find a point r such that the plane that contains p, q and r is above all current points.
2. Remove r from the set of current points and set p = q and q = r.

The final output is the reverse of the order the points were taken from s, with p and q at the end.

To do the first step above (find r), we could just try every possible point in s and use the process described above to check the condition. This would yield an O($\mathbf{N}^2$) algorithm for the step. To improve upon that, picture the plane containing the last roof, but now being supported by the remaining 2 fixed columns. For the very first iteration, the "last roof" needs to exist for this

algorithm to be valid, and how to define it depends on how we choose p and q. For the choice of p and q we explain below, the existence of a possible "last roof" is straightforward. We can rotate the plane around the axis defined by the segment connecting the tips of those two columns, and we want to find one of the columns that it touches first. (There can be up to two such columns, since we can rotate the plane in either direction.) In other words, we should choose a column tip r that minimizes the angle of rotation. That is equivalent to choosing the r that maximizes the angle between a normal to the plane that contains p, q and r and the normal to the "last roof". This yields a linear time algorithm for this step.

To choose the initial p and q, we can try every possibility, which adds an $\mathbf{N}^2$ factor to the overall time complexity of the algorithm, or do something better. The last column with tip p can always be chosen as one of the tallest columns. Imagine a plane parallel to z = 0 that passes through p. The plane clearly passes above or touches all other columns, but it's not "pierced" by any column, so we can move it around to create a suitable roof. Then, we can choose q as a column such that the segment pq has minimal angle with the plane. This is similar to what we did above, except we are trying rotating the plane around a single point p until it touches another column q. This can be done by using the same method of comparing angles against a plane we described in the previous paragraph. This involves a one-time linear cost to find q, but then we only do the iteration for a single pair. Notice that, for this choice of p and q, there is a clear choice for the "last roof", which is the imaginary plane that we rotated to find q.

The optimal implementation of the ideas above yields a quadratic algorithm, which is the intended solution. Depending on which of the optimizations you skip, you can end up with a complexity between $O(\mathbf{N}^2)$ and $O(\mathbf{N}^5)$, which, together with the constant factors of your implementation and choice of language, determines whether your solution passes both sets, just Test set 1, or neither set.