Solution for Comparing Plants

For simplicity of describing the solutions, all indices of the array r will be taken modulo n.

Subtask 1

We are given the height comparisons of each pair of adjacent plants. We can conclude that plant x is taller than plant y if either of the following holds:

- $r[x] = r[x+1] = \ldots r[y-1] = 0$
- $r[y-1] = \ldots = r[n-2] = r[n-1] = 1$ and $r[0] = r[1] = \ldots = r[x-1] = 1$

The condition for returning -1 is similar.

If none of the conditions hold, return 0.

Implementing this naively requires $O(nq)$ time. We can speed this up by using a static prefix sum array.

Subtask 2

The overall idea is to generate a possible configuration of heights.

First observe that the tallest plant x must satisfy $r[x] = 0$. This suggests the following algorithm:

- Repeat the following for n times:
  1. Look for some index x such that $r[x] = 0$
  2. Decrement $r[x-k+1], r[x-k+2], \ldots, r[x-1], r[x]$ by 1

Consider the input $n = 3, k = 2, r = [0, 1, 0]$. The tallest plant must be 2. However, we also have $r[x] = 0$. We therefore need to replace rule (1) with the following:

- Look for some index x such that $r[x] = 0$ and $r[x-k+1], r[x-k+2], \ldots, r[x-k+1] \neq 0$. (*)

The index x satisfying the above condition is always unique (and therefore the final configuration of heights is also unique). The order in which these x are chosen will give rise to a possible configuration of heights, in which we can answer all the queries by simply comparing the heights. Complexity $O(n^2)$.

Subtask 3

Same as above; except that we need to use a segment tree with lazy propagation to speed up the algorithm. However, it can be tricky to implement (*).

To do this, we choose a value t such that $r[t] = 0$. If there are multiple of them, choose the smallest. We can do a range minimum query to check if any of $r[t-k+1], \ldots r[t-1]$ is zero. If none of them are zero, we choose $x = t$.

If not, we let x be the smallest index among $t-k+1, \ldots, t-1$ such that $r[x] = 0$.

Other than the potentially more tedious implementation of (*), the rest of the algorithm remains the same. Complexity $O(n \log n)$.

Subtask 4

The implementation of (*) becomes more problematic here. To illustrate, suppose that $k = 2$ and $r[0] = r[1] = \ldots = r[n-2] = 0, r[n-1] = 1$.

If we choose the value $t = n-2$, we realise that $r[n-3] = 0$. However, we run into further problems because $r[n-4]$ is also zero.

As such, we need to recursively search for the correct index. Complexity $O(n \log n)$.

```
extract(x):
  while there is a y in {x-k+1, x-k+2, ..., x-1} such that r[y] == 0:
    extract(y)
  end
  decrement r[x-k+1], r[x-k+2], ..., r[x] by 1
```

The ordering of the heights is then given by the order in which the procedure returns (rather than the order the of being called). The queries can still be answered by a direct comparison of the heights.

Subtask 5

We first generate a possible configuration of heights $h[0, 1, \ldots n-1]$ using the techniques from the previous subtasks.

Define $d(x, y) = \min(|x-y|, |x+n-y|, |y+n-x|)$ to be the distance between plants x and y. Generate a directed graph with $x \to y$ if and only if $h[x] > h[y]$ and $d(x, y) < k$.

Lemma:

- The above graph generated does not depend on the choice of $h[]$. In other words, any two height arrays $h_1[]$ and $h_2[]$ consistent with $r[]$ will generate the same directed graph.

Proof:

- Let G be any directed graph consistent with the input $r[]$ (that is, for each i, there are exactly

$r[i]$ values of j among $i+1, i+2, \ldots i+k-1$ such that $j \rightarrow i$).

- We first observe that any valid configuration $h[]$ corresponds to a run of the algorithm (*).
- It is therefore sufficient to show that any run of the algorithm (*) generates a topological sort of this graph G.
- The proof is similar to Kahn's algorithm. When plant i is removed by the algorithm (*), the conditions for removing i guarantee that i must have zero in-degree.
- Since we only remove nodes with zero in-degree, the order generated must correspond to a valid topological sort of G.

To check if plant x is taller than plant y, we check if there exists a (directed) path from x to y.

To do so, we can compute the transitive closure using Flyod-Warshall and pre-compute the answers to the entire space of queries. Complexity $O(n^3)$.

Alternative solution to subtask 5

We follow the idea of (*). We look for some z such that $r[z]=0$ and $r[z-k+1], r[z-k+1], \ldots, r[z-k+1] \neq 0$, but now including an additional constraint that $z \neq x$. We stop when x is the only plant we can remove.

There exists a configuration of heights such that plant y is taller than plant x if and only if the above algorithm removes plant x.

Take note that doing so naively would require $O(qn^2) = O(n^4)$ computation time, if we do not use a segment tree.

However, it is easy to speed this up to $O(n^3)$. For each plant $0 \leq x \leq n-1$, we pre-compute in advance all y such that y can possibly be taller than x, storing them in a $n \times n$ table.

Subtask 6

Same idea as subtask 5. However, we are unable to generate the entire graph as there can be up to $O(nk)$ edges.

For each plant x, define the 'left edge' and 'right edge' as follows:

- $left(x) =$ the tallest plant shorter than x among $x-k+1, \ldots, x-1$, or -1 if it does not exist
- $right(x) =$ the tallest plant shorter than x among $x+1, \ldots, x-k+1$, or -1 if it does not exist

We now generate a graph with edges $x \rightarrow left(x)$ and $x \rightarrow right(x)$ for each x. To check if plant x is taller than plant y, we check if there exists a (directed) path from x to y.

Since we only need to compare against plant 0, we can use single-source BFS or DFS to find all vertices reachable from 0.

Complexity $O(n \log n)$ (the queries are answered in $O(1)$ time, however we need $O(n \log n)$ to generate the array $h[]$, each value of $left(x)$ and $right(x)$ requires $O(\log n)$ to compute.

We may also use the idea to the alternative solution of subtask 5. After finding all the plants that can be taller than plant 0, we can invert the ranks (i.e. replace $r[i]$ with $k-1-r[i]$ for each i) to check if a plant can shorter than plant 0.

Subtask 7

For each query (x, y), let z be the first value among $left(x), left(left(x)), left(left(left(x))), \ldots$ such that y is between z and $left(z)$.

If $h[z] > h[y]$, we can then conclude that x is taller than y.

We do so similarly for the right edges.

To speed up this computation, we use power-of-2 tables:

$$left_1(x) = left(x), left_{k+1}(x) = left_k(left_k(x))$$

Using this approach, we may compute z using $O(\log n)$ time.

Total complexity: $O(n \log n + q \log n)$.

To the readers who are keen for an extra challenge: figure out how to solve this problem using $O(n \log n)$ preprocessing time and $O(1)$ time per query!