Solution for Teleporters

A series of N teleporters with 2N distinct endpoints divides the line into 2N+1 intervals. Since nothing interesting can happen to you while traversing one of these intervals, traversing one can be considered instanteneous.

From each of these intervals except the last one, we can draw an arrow to the next interval we would visit after traversing this interval and getting teleported. A few quick observations can now be made:

- Each arrow corresponds to an end of a teleport.
- There are exactly 2N arrows.
- Each interval except the first and last has exactly one arrow going out of it and one coming into it.
- The first interval only has an arrow going out of it.
- The last one only has one coming into it.

Consider the graph where intervals are vertices and our arrows are edges. This graph may consist of several components. It is obvious that one of them is always a path (and includes the first and the last interval). The other components, if present, must always be cycles.

When we add a new device, we are essentially cutting up some intervals and 'rewiring' the edges. There are 3 cases to consider when adding the device, and it's not very difficult to verify by casework what happens in each of these cases:

Case 1: The endpoints of the new device are in two intervals that are not in the same component.
These two components are merged into one. The number of edges in the new component is two more than the total number of edges in both original components.

Case 2: The endpoints of the new device are in two intervals on the same component C.
Let X be the length of the path from the first to the second interval, i.e., the number of jumps we currently have to make in order to reach the second interval from the first one. By adding the new bypass, we will remove these X jumps from the component C and replace them by a single jump using the new teleporting device. Thus the number of edges in C decreases by X-1. Additionally, we get a new cycle with X+1 edges.

Case 3: The endpoints of the new device are in the same interval.
The current component has its number of edges increased by 1, and another cycle of lenght 1 is created.

We can now use a greedy approach to find the best placement for the M new devices. If there are at least M cycles, it is clearly optimal to take the M largest cycles and connect them to the path (by placing the devices according to case 1 above). If we get to the situation where no cycles remain, we have to place the next device according to case 3. In this way, we get a new cycle of length 1.

The components can be detected by a simple breadth first search. Then this process can be simulated in O(N log N + M), for example by sorting the cycle sizes, or by using a priority queue. It is possible to improve the time complexity to O(N log N) by handling the case when no more cycles are left in constant time. Such a solution was expected to achieve the full score. Furthermore, counting sort can be used to sort cycle sizes to lower the time complexity to O(N).