Application of sets

This problem is a simple application of the set data structure. The first part is from the ICPC world finals 2019. The actual description of the problem can be found at https://icpc.kattis.com/problems/azulejos. A more formal description is given below.

Let S_1 and S_2 be two sequences of length n, the elements of which are ordered pairs of integers, not necessarily distinct. Let $S_1 = (a_1, b_1), \ldots, (a_n, b_n)$ and $S_2 = (c_1, d_1), \ldots, (c_n, d_n)$. The problem is to find permutations p and q of $\{1, \ldots, n\}$, if they exist, satisfying the following properties.

1. $a_{p(i)} \leq a_{p(i+1)}$ and $c_{q(i)} \leq c_{q(i+1)}$ for $1 \leq i < n$.
2. $d_{q(i)} \leq b_{p(i)}$ for $1 \leq i \leq n$.

In other words, the elements in S_1 and S_2 must be ordered in non-decreasing order of their first coordinate, such that the ith element in the ordering of S_2 has second coordinate less than or equal to the second coordinate of the ith element in the ordering of S_1. Such orderings may not always be possible.

The basic algorithm is as follows. Let a be the minimum value of the a_i and c the minimum of the c_i. Let A be the subset of pairs in S_1 whose first coordinate is equal to a, and C the subset of pairs in S_2 with first coordinate c. The elements in A must come first in the ordering of S_1 and the elements in C must come first in the ordering of S_2. Suppose $|A| \leq |C|$. Then for every element of A the corresponding element in the ordering of S_2 must be an element in C. Choose an element in C such that its second coordinate is the largest possible but does not exceed the second coordinate of the element in A. Delete these elements from A and C. If for some element in A, no such element in C is found, then an ordering is not possible. If A becomes empty, some elements may be left in C. Now consider the second smallest value of the first coordinate in S_1, let A be the set of those elements and repeat. If $|C| \leq |A|$, do the same for elements in C, except that the element of A chosen is the one with smallest second coordinate that is greater than or equal to the second coordinate of the element in C. The lower_bound and/or upper_bound methods of the set class can be used to find these elements.

The second part of the problem, which was not in the ICPC, is to determine whether the solution is unique, that is, there exists exactly one pair of permutations p, q that satisfies the given property.

Input/Output: The first line of input specifies n, which will be at most 5×10^5. The next 4 lines contain n numbers each separated by a space, with each number at most 10^9. The first line gives the values of a_i, the next b_i, c_i, d_i in that order. If there exist permutations p, q satisfying the required properties, print p on the first line and q on the second. If there are many solutions, any one is okay. Note that indices are considered from 1 to n. If there are no such permutations, print "impossible", without quotes. For the second part, if there exist such permutations, print "unique" if the answer is unique, otherwise print "not unique". Nothing needs to be done if it is impossible. Time limit in the contest was 10sec but it should probably take less. The test cases from the contest,

for the first part, are put up on teams. I may use different ones, especially for the second part.

Submission: Submit a single file named RollNo_9.cpp

Note: It may be necessary to compare ordered pairs of numbers based on the first coordinate, or the second coordinate at different times. (It may in fact be easier to consider triples, where the third coordinate is the index). By default, sets and functions like sort use the < operator for comparison. If a different comparison function is required, it can be specified when defining the set or calling the sort function. This is done using function objects, for which the function call operator () is defined. A sample declaration would be

```
struct compare{
     bool operator()(T const &t1, T const &t2) const
     {
          // comparison for elements of type T
     }
};

sort(v.begin(), v.end(), compare());

// v is a vector of type T, a dummy object of type compare is passed
// to the sort function. If f is an object of type compare, f(t1,t2)
// returns a boolean value, comparing t1 with t2.

set<T,compare> S;

// defines a set with elements of type T using compare
// as the comparison operation.
```

```
Sample Input 1 Sample Output 1
4              3 2 4 1
3 2 1 2        4 2 1 3
2 3 4 3        not unique
2 1 2 1
2 2 1 3

Sample Input 2 Sample Output 2
2              1 2
1 2            1 2
2 3            unique
2 8
2 1
```