Annotation:

1. It is good to think of the idea of first sorting the start milking time and then using greedy algorithm.

2. Using merge sort is not bad, but after sort each start milking time, we have to permute its relative end time array to match, which will take much running time. (fail to finish within allow time: 1s)

3. Therefore, the answer.java use Arrays.sort() instead to sort the Interval object (class define by user to maintain both start and end milking time)

4. How to implement Arrays.sort()?

1) sort() is static method, thus could be used like: Arrays.sort(T[] a, Comparator<? super T> c). Sorts the specified array of objects according to the order induced by the specified comparator.

2)
```
Arrays.sort(intervals, new Comparator<Interval>() {
    //define a class implement Comparator interface
    //override compare() method
    @Override
    public int compare(Interval o1, Interval o2) {
        return o1.getLow()-o2.getLow();
    }
});
```

3. Strong suggest using Arrays.sort(MyObject, Comparator<? super T>), it is clear to understand. Plus, if there are several related array, like start[], processing[], end[], you can package them into a three item class and use Array.sort to permute.