Gridcity, a small and peaceful city, is well known for the coexistence of cats and people and its perfectly rectangular urban plan.

Since cats are serious about social distancing these days, there are frequent conflicts between cats due to overlapping territories. Gridcity's cats are so independent that they are ready to fight whenever the area of intersection of any two cats' territories is positive.

Meanwhile, as more people couldn't sleep because of the fights, the mayor of Gridcity decided to select some of the cats and move them to neighboring cities. Not surprisingly, the cats were opposed to leaving their hometown. After lengthy discussions, the mayor and cats agreed that only a minimal number of cats have to be moved, and one leader cat is always guaranteed to remain in Gridcity.

Suppose that Gridcity has width W and height H, and the lower-left and upper-right coordinate of Gridcity is $(0, 0)$ and (W, H), respectively. There are n cats from index 1 to n (inclusive), and the ith cat's territory is a rectangle whose lower-left corner is at $(A_i, 0)$, and upper-right corner is at (B_i, H), and $0 \leq A_i < B_i \leq W$. Cats are sorted by A_i in ascending order, and there are no duplicate coordinates. The leader cat is at index k, and $1 \leq k \leq n$. Coordinate values, lengths, and indices are all integers.

(1) [10 points] Find an $O(n)$ algorithm that returns a list of cats to move. Of course, your algorithm should comply with the agreement between cats and people. Write your algorithm in English, Korean, or pseudocode. You don't have to prove correctness.

Input: n k W H $A_{1\cdots n}$ $B_{1\cdots n}$ *Output*: an array of indices

(2) [5 points] Show that your algorithm runs in $O(n)$ time.

Solution and grading policy for (1)

The Gridcity problem can be solved by using greedy scheduling algorithm, but the leader cat and the ordering of cats must be considered for correct output.

1 Greedy scheduling method **[4 pts]**

If you use greedy approach which is similar to the algorithm for scheduling problem and find the maximum remaining cats in the Gridcity, you will get full credit.

2 Leader cat **[2 pts]**

Since leader cat must be remained in the Gridcity, every other cats overlapped with the leader cannot be remained. Notice that any of the cats can be overlapped with leader even though they are already sorted, so their indices doesn't matter and you need to check whether (A_i, B_i) and (A_k, B_k) intersect or not for all i^{th} cats. If your algorithm successfully excludes the cats matched, you will get full credit.

3 Searching order **[2 pts]**

Unlike the original scheduling problem, the cats are sorted by A_i in increasing order, instead of B_i. This condition requires that the cats must be searched in reverse order of A_i (i.e., n^{th} cat is remained, $(n-1)^{th}$ cat should be handled next (remained or not, according to overlapping condition), and so on.) If the reversed searching order is correctly and explicitly explained, you will get full credit (no credit if you don't mention it too).

4 $O(n)$ algorithm **[2 pts]**

If the time-complexity of your algorithm is $O(n)$, you get full credits.

Common mistake - Sorting again by B_i takes $\Omega(nlogn)$ (you may argue that it can be $O(n)$, but it is not since W and H are not constants and totally independent to n), so additional sorting cannot be possible for this problem.

Solution and grading policy for (2)

1 $O(n)$ algorithm **[2 pts]**

If you give $O(n)$ time-complexity algorithm, you get full credit.

2 Correct explanation for $O(n)$ **[3 pts]**

If you correctly prove that it is $O(n)$ time-complexity algorithm, you will get full credit.