Q1:

For the first question, in fact, I fell into a misunderstanding when solving the problem at the beginning. The easiest and most violent way to do this, of course, is to go through every player and compare them to everyone else in the loop. But obviously, the time complexity would be enormous. So I started thinking about whether I could simplify the process. The first thing that came to my mind was to divide the players into two groups, based on U and D, and queue both groups. Then traverse the two queues and compare. When a queue has completely traversed once, it ends the current loop and enters the next one. Until the last two queues are empty, or there are elements in the queue but no more encounters are possible. The biggest problem with this solution is that the outermost while loop, which says the player keeps moving, is unnecessary. This is because each player moves at the same speed, and the player state(hp) does not change during the movement. So we can compare a player from start to finish. So, that's my final solution. First, the input player is sorted by their starting floor, and then an empty stack is created. Compare the current player outside the stack with the player at the top of the stack, if the former is D and the latter is U, then continue to compare their hp, and then decide who will be pushed in the stack and who will be popped out of the stack. It is worth noting that if the former head is removed from the stack, the outside player will continue to be compared with the next of the head until the stack is empty or the outside element is no longer larger than the top element. In this way, we reduce the cycle to one time, greatly reducing the time complexity. Finally, the elements in the stack are iterated through, sorted in order of input, and finally output.

Q2:

For the second problem, the simplest and most violent approach is to extend for each depth separately by traversing to the left and right until the left and right boundary is found, that is, the first value less than that depth. But obviously, the time complexity of this method is too high. To solve this problem, we can first consider a very special case, that is, the depth is arranged in an increasing order. For such cases, we do not need to traverse the boundary when finding the area, but directly determine the left boundary by the depth itself, and then the right boundary is the last depth. So can we adjust this disordered depth combination into an orderly way? We can do this using a monotonic stack. For the input depth, we start from the first one in depth combination, and compare it with the top of the stack element, when the outside deep is greater than the head, we directly push it into the stack, and another situation, we pop the top of the stack element out. After that, firstly we calculate the depth of the pop-out can reach the maximum area, and then continue to let the current outside the stack element and the next top of the stack element compare, until the stack is empty or the outside of the stack element greater than the top of the stack element. At the end of this traversal, we will have a deep combination decreasing from the top of the stack to the bottom, using the way we just imagined for ordered combinations, to find the maximum value. In this way, the time complexity of the method can be O(N) compared to multiple traversals.