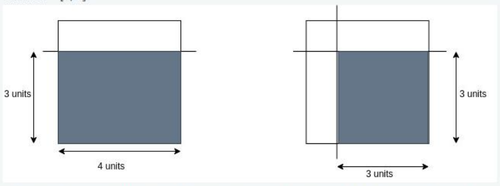
1. Largest Area

Given a rectangle, add vertical and horizontal separators at various locations. Determine the area of the largest open space after each line is added. Return these values in an array.

Example

w = 4
h = 4
isVertical = [0, 1]
distance = [3, 1]



The values in *isVertical* indicate whether a boundary is horizontal (*isVertical[i] = 0*) or vertical (*isVertical[i] = 1*),
The values in *distance* are the distances from the boundary at 0 in either direction. In the graphics above, that is from the bottom if the boundary is horizontal and from the left if is is vertical.

In the first graph, a horizontal line (*isVertical[0] = 0*) is created *distance[0] = 3* units above the bottom. This creates two areas of *1 x 4 = 4* and *3 x 4 = 12* units each. The largest area is *12.*

In the second graph, a vertical line (*isVertical[1] = 1*) is created at *distance[1] = 1* unit from the left. There are now four areas, the largest of which is *3 x 3 = 9* units in size.

The return array is *[12, 9].*

Note: The horizontal and vertical lines may not be distinct. Placing a new boundary at a previous boundary location does not change the graph.

Function Description
Complete the function *getMaxArea* in the editor below.

getMaxArea has the following parameters:
int w: the width of the rectangle
int h: the height of the rectangle
bool isVertical[n]: 0 denotes a horizontal boundary and 1 denotes a vertical boundary
int distance[n]: the distance to each boundary, either from the bottom or from the left of the rectangle

Returns:
int[n]: each element *i* is the size of the largest open area after adding boundary *i*

Constraints

- $2 \le w, h \le 10^5$
- $1 \le n \le 10^5$
- It is guaranteed that *isVertical[i]* is either 0 or 1.
- $1 \le distance[i] \le w\text{-}1$ for a vertical boundary
- $1 \le distance[i] \le h\text{-}1$ for a horizontal boundary

▼ Input Format For Custom Testing

The first line contains an integer, *w*, the width of the rectangle.
The second line contains an integer, h, the height of the rectangle.
The third line contains an integer, *n*, the size of the *isVertical* array.
Each line *i* of the *n* subsequent lines (where $1 \le i \le n$) contains either 0 or 1, corresponding to *isVertical[i].*
The next line contains an integer, *n*, the size of the *distance* array.
Each line *i* of the *n* subsequent lines (where $1 \le i \le n$) contains an integer, *distance[i].*

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     Function
-----     --------
2     →   w = 2
2     →   h = 2
2     →   isVertical[] size n = 2
0     →   isVertical = [0, 1]
1
2     →   distance[] size n = 2
1     →   distance = [1, 1]
1
```

Sample Output

```
2
1
```

Explanation
The rectangle is *2 x 2* units size.
The first boundary is horizontal at distance 1 from the bottom. It creates 2 smaller rectangles with areas of *2 x 1 = 2* units each.

▼ Sample Case 1

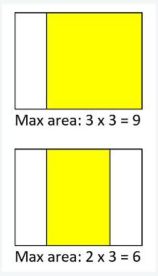
Sample Input For Custom Testing

```
STDIN    Function
-----    --------
4    →   w = 4
3    →   h = 3
2    →   isVertical[] size n = 2
1    →   isVertical = [1, 1]
1
2    →   distance[] size n = 2
1    →   distance = [1, 3]
3
```

Sample Output

```
9
6
```

Explanation
The rectangle is *4 x 3* units size.
first boundary is a vertical boundary made at a distance of 1 unit from the left. This creates 2 smaller rectangles of *1 x 3 = 3* units and *3 x 3 = 9* units.
The second boundary is vertical at position 3 from the left. This leaves 3 smaller rectangles, 2 with areas *1 x 3 = 3* units and 1 with an area of *2 x 3 = 6* units.


Max area: 3 x 3 = 9
Max area: 2 x 3 = 6