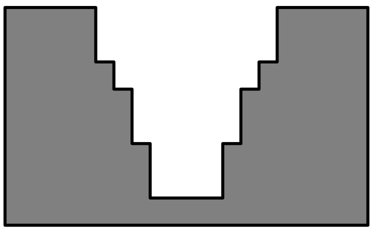
Falling Disks

This time we will see how the Codility Certificate codenamed Ω (Omega) can be solved. You can still give it a try, but no certificate will be granted. The story is about throwing concrete disks into an old dry well. The well is made of concrete rings that can have different (internal) diameters. The disks fall until they hit the bottom, the previously dropped disk or the concrete ring (whose internal diameter is smaller then the disk's diameter). You are given two arrays, A containing the diameters of the rings in the top-down order, and B containing the the diameters of the disks. The task is to compute how many disks will fit into the well?

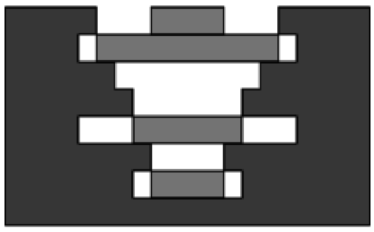


A simple solution is to simulate the falling disks.

1: Simple solution — $O(N \cdot M)$.

```
def falling_disks(A, B):
    N = len(A)
    M = len(B)
    i = 0
    while i < M and N > 0:
        k = -1
        while k + 1 < N and B[i] <= A[k + 1]:
            k = k + 1
        if k >= 0:
            i = i + 1
        N = k
    return i
```

However, it is inefficient. Simulation of one disk can take $O(N)$ time, and the whole algorithm can require $O(N \cdot M)$ time. To find a faster solution, we will simplify the problem first. Imagine that you are looking into the well from above. It seems that the rings that are deeper are smaller. Even if one of them in fact has a larger diameter, it is hidden behind a narrower ring above. We can apply such a transformation to the input data without changing the result. Note that a disk can reach some ring if it is not wider then the diameter of this ring and all the rings above. For the example shown on the figure above we obtain the following well:



Such a transformation can be computed in $O(N)$ time. Then, we can find the final position of each disk by checking the rings in the bottom-up order. Since it is not possible to fit more disks than there are rings, the overall time complexity is $O(N)$.

2: Model solution — $O(N)$.

```
def falling_disks(A, B):
    N = len(A)
    M = len(B)

    for i in range(1, N):
        A[i] = min(A[i], A[i-1])

    i = 0
    while i < M and N > 0:
        while N > 0 and A[N-1] < B[i]:
            N = N - 1
        if N > 0:
            N = N - 1
            i = i + 1

    return i
```