An arcade game player wants to climb to the top of the leaderboard and track their ranking. The game uses Dense Ranking, so its leaderboard works like this:

- The player with the highest score is ranked number 1 on the leaderboard.
- Players who have equal scores receive the same ranking number, and the next player(s) receive the immediately following ranking number.

Example

$ranked = [100, 90, 90, 80]$
$player = [70, 80, 105]$

The ranked players will have ranks 1, 2, 2, and 3, respectively. If the player's scores are 70, 80 and 105, their rankings after each game are 4^{th}, 3^{rd} and 1^{st}. Return $[4, 3, 1]$.

Function Description

Complete the climbingLeaderboard function in the editor below.

climbingLeaderboard has the following parameter(s):

- int ranked[n]: the leaderboard scores
- int player[m]: the player's scores

Returns

- int[m]: the player's rank after each new score

Input Format

The first line contains an integer n, the number of players on the leaderboard.
The next line contains n space-separated integers $ranked[i]$, the leaderboard scores in decreasing order.
The next line contains an integer, m, the number games the player plays.
The last line contains m space-separated integers $player[j]$, the game scores.

Constraints

- $1 \leq n \leq 2 \times 10^5$
- $1 \leq m \leq 2 \times 10^5$
- $0 \leq ranked[i] \leq 10^9$ for $0 \leq i < n$
- $0 \leq player[j] \leq 10^9$ for $0 \leq j < m$
- The existing leaderboard, $ranked$, is in descending order.
- The player's scores, $player$, are in ascending order.

Subtask

For 60% of the maximum score:

- $1 \leq n \leq 200$
- $1 \leq m \leq 200$

Sample Input 1

Copy Download

```
7
100 100 50 40 40 20 10
4
5 25 50 120
```

Sample Output 1

```
6
4
2
1
```

Sample Input 2

Copy Download

```
6
100 90 90 80 75 60
5
```

```python
# The function is expected to return an INTEGER_ARRAY.
# The function accepts following parameters:
#  1. INTEGER_ARRAY ranked
#  2. INTEGER_ARRAY player
#

def climbingLeaderboard(ranked, player):
    # removing duplicates from the ranked list
    ordered_map = {}
    for rank in ranked:
        if rank not in ordered_map: ordered_map[rank] = True

    # duplicates have been removed, get back the list
    ranked = list(ordered_map.keys())
    stack = []

    # resultant ranks of all the players according to their scores in list player
    result = []

    # this will help us not append the scores back into the ranked list from the stack
    player.sort()

    # assuming current score's rank to be the last, and we will update it as we compare it with scores in stack
    counter = len(ranked) + 1

    for score in player:

        while len(ranked) != 0:
            popped = ranked.pop()

            if popped > score:
                result.append(counter)

                # for next comparison
                ranked.append(popped)
                # rank for this score has been determined break out of the loop
                break

            counter -= 1

        # when the while loop will determine that the current element's rank is 1
        # the ranked list will get empty and the rank will not be appended
        # hence, this check is necessary
        if counter == 1: result.append(1)

    return result


if __name__ == '__main__':…
```

Line: 60 Col: 27

Upload Code as File   Test against custom input   Run Code   Submit Code

Congratulations

You solved this challenge. Would you like to challenge your friends?