Alice está jugando un juego de arcade y quiere subir a la cima de la tabla de clasificación y quiere seguir su clasificación. El juego usa Dense Ranking, por lo que su tabla de clasificación funciona así:

- El jugador con el puntaje más alto es el número clasificado en la tabla de clasificación.
- Los jugadores que tienen puntajes iguales reciben el mismo número de clasificación, y los siguientes jugadores reciben el número de clasificación inmediatamente siguiente.

Por ejemplo, los cuatro jugadores en la tabla de clasificación tienen puntajes altos de 100,90,90 y 80. Esos jugadores tendrán rangos 1, 2, 2 y 3, respectivamente. Si los puntajes de Alice son 70, 80 y 105, sus clasificaciones después de cada juego son 4°, 3° y 1°.

Función descriptiva

El problema consiste en completar la función climbingLeaderboard la cual deberá devolver una estructura de datos donde cada elemento representa los rankings de Alice después de completar una cantidad "j" de juegos.

La función climbingLeaderboard tiene los siguientes parámetros:

· int scores_count: cantidad de jugadores en la clasificación

· int* scores: una serie de enteros que representan los puntajes de la tabla de clasificación

· int alice_count: cantidad de partidos jugados por Alice

· int* alice: serie de enteros que representan las puntuaciones de Alice.

· (xxx) result_count: es la estructura de datos de salida. NOTA: Se deberá elegir el tipo y estructura de datos usar.

Formato de entrada

- La primera línea contiene un número entero n, el número de jugadores en la tabla de clasificación.
- La siguiente línea contiene n enteros separados por espacios scores[i], la tabla de clasificación puntúa en orden decreciente.
- La siguiente línea contiene un número entero m, que denota los juegos de números que juega Alice.
- La última línea contiene m enteros separados por espacios alice[j], los puntajes del juego.

Constraints

- 1 <= n <=2 x 105
- 1 <= m <=2 x 105
- 0 <= scores[i] <= 109 for 0 <= i < n
- 0 <= alice[j] <= 109 for 0 <= j < m
- La tabla de clasificación existente, scores[], está en orden descendente.
- Las puntuaciones de Alice, alice[], están en orden ascendente

Subtarea

Para 60% de la puntuación máxima:

- 1 <= n <= 200
- 1 <= m <= 200

Formato de salida

Imprimir m enteros. El j° número entero debe indicar el rango de Alice después de jugar el juego j°

Ejemplo de entrada

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

Ejemplo de salida

```
6
4
2
1
```

Explicacion 1

Alice comienza a jugar con 7 jugadores que ya están en la clasificación, que se ve así:

Rank	Name	Score
1	Emma	100
1	David	100
2	Caroline	50
3	Ritika	40
3	Tom	40
4	Heraldo	20
5	Riley	10

Después de que Alice termina el juego 0, su puntaje es 5 y su clasificación es 6:

Rank	Name	Score
1	Emma	100
1	David	100
2	Caroline	50
3	Ritika	40
3	Tom	40
4	Heraldo	20
5	Riley	10
6	**Alice**	**5**

Después de que Alice termina el juego 1, su puntaje es 25 y su clasificación es 4:

Rank	Name	Score
1	Emma	100
1	David	100
2	Caroline	50
3	Ritika	40
3	Tom	40
4	**Alice**	**25**
5	Heraldo	20
6	Riley	10

Después de que Alice termina el juego 2, su puntaje es 50 y su clasificación está vinculada con Caroline en 2:

Rank	Name	Score
1	**Alice**	**120**
2	Emma	100
2	David	100
3	Caroline	50
4	Ritika	40
4	Tom	40
5	Heraldo	20
6	Riley	10

Después de que Alice termina el juego 3, su puntaje es 120 y su clasificación es 1:

Rank	Name	Score
1	**Alice**	**120**
2	Emma	100
2	David	100
3	Caroline	50
4	Ritika	40
4	Tom	40
5	Heraldo	20
6	Riley	10

```
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.regex.*;

public class Solution {

// Complete la siguiente funcion climbingLeaderboard.
static int[] climbingLeaderboard(int[] scores, int[] alice) {


}

private static final Scanner scanner = new Scanner(System.in);

public static void main(String[] args) throws IOException {
BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

int scoresCount = scanner.nextInt();
scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

int[] scores = new int[scoresCount];

String[] scoresItems = scanner.nextLine().split(" ");
scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");
```

```
for (int i = 0; i < scoresCount; i++) {
int scoresItem = Integer.parseInt(scoresItems[i]);
scores[i] = scoresItem;
}

int aliceCount = scanner.nextInt();
scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

int[] alice = new int[aliceCount];

String[] aliceItems = scanner.nextLine().split(" ");
scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

for (int i = 0; i < aliceCount; i++) {
int aliceItem = Integer.parseInt(aliceItems[i]);
alice[i] = aliceItem;
}

int[] result = climbingLeaderboard(scores, alice);

for (int i = 0; i < result.length; i++) {
bufferedWriter.write(String.valueOf(result[i]));

if (i != result.length - 1) {
bufferedWriter.write("\n");
}
}

bufferedWriter.newLine();

bufferedWriter.close();

scanner.close();
}
}
```