```
REVISION HISTORY:
2022-11-13: Original
```

pa08a

Problem description

Deven and Jake collect Digimon cards. Digimon, short for "Digital Monster", are mysterious lifeforms that were discovered in the Computer Network in 1997. Digimon include almost all the creatures that reside within a parallel universe called the "Digital World." There are many different species of Digimon, as well as unconfirmed and new generations of Digimon yet to be revealed. The number of Digimon species continuously increases as the Digital World expands each year. One card is dedicated to each Digimon species. To make it easier, we'll assume each Digimon card has a unique identifier, given as an integer number.

Both Deven and Jake have a set of cards and like to trade the cards they have. They obviously don't care about identical cards they both have, and they don't want to receive repeated cards in the trade. Besides, the cards are traded in a single operation: Deven gives Jake *N* distinct cards and receives back another *N* distinct cards. The guys want to know: what is the maximum number of cards they can trade? For instance, if Deven has cards {1, 1, 2, 3, 5, 7, 8, 8, 9, 15} and Jake has cards {2, 2, 2, 3, 4, 6, 10, 11, 11}, they can trade at most four cards.

Write a modular C++ program that, given the sets of cards owned by Deven and Jake, determines the maximum number of cards they can trade.

Input specification

The input contains several test cases. The first line of a test case contains two integers $\boldsymbol{A}$ and $\boldsymbol{B}$, separated by a blank space, indicating respectively the number of cards Deven and Jake have ($1 \le \boldsymbol{A} \le 10^4$ and $1 \le \boldsymbol{B} \le 10^4$). The second line contains $\boldsymbol{A}$ space-separated integers $\boldsymbol{X}_i$, each indicating one of Deven's cards ($1 \le \boldsymbol{X}_i \le 10^5$). The third line contains $\boldsymbol{B}$ integers $\boldsymbol{Y}_i$ separated by whitespaces, each number indicating one of Jake's cards ($1 \le \boldsymbol{Y}_i \le 10^5$). Deven and Jake's cards are listed in non-descending order.

The end of input is indicated by a line containing only two zeros, separated by a blank space.

Output specification

For each test case, your program should print a single line containing an integer value indicating the maximum number of cards Deven and Jake can trade.

Sample input

```
1 1
1000
1000
3 4
1 3 5
2 4 6 8
10 9
1 1 2 3 5 7 8 8 9 15
2 2 2 3 4 6 10 11 11
0 0
```

Sample output

```
0
3
4
```

Specific program requirements

- This program must be designed modularly, i.e., using functions. The `main()` function body should not exceed 25 lines of code.
- Implement and use the following functions in your solution:

  ```
  /// @brief Returns index of the first element in the array where an
  /// element is equal to the target value. If the target value is not found,
  /// returns size.
  ```

```
///
/// @param list The array to search.
/// @param size The number of elements in the array.
/// @param target The search target value.
/// @returns The index of the first occurrence of target in list, or,
/// size if not found.

unsigned find(const int list[], unsigned size, int target);
```

```
/// @brief Reads up to size values into list from the standard input stream.
/// Only unique values are stored in the array (i.e., no duplicates). Returns
/// the number of unique values read into the array.
///
/// @param list The array to initialize.
/// @param size The number of elements in the array.
/// @returns Number of unique values read into the array.

unsigned read(int list[], unsigned size);
```

- You may want to write more functions to support your implementation and reduce the size of your main() function.

> Hint: *During development* it's frequently handy to print out an array for visual inspection. Don't work in the dark.

Check script

You will find a self-check script (`/home/shared/cs135/kmess/pa08a/pa08a-check`) to diagnose and test your program (Available Thursday.) Until then, remember to check your code with cpplint, cppcheck, and/or pclint tools (in addition to using $CXXFLAGS when compiling at the command line).

> Static analysis tools do not understand our programs. They can only detect things that are demonstrably wrong or are otherwise suspicious. It's up to the developer (us) to determine if a particular diagnostic applies or not. Strive for as few static analysis diagnostics as possible. It's a good thing.

However,

> Your final program should *compile* cleanly without errors or warnings when using the $CXXFLAGS set of options.

The Judge (TM)

You will find several test inputs and expected outputs in the `/home/shared/cs135/kmess/pa08a/` subdirectory. After copying these to your project directory you can type, for example:

```
$ judge -p ./pa08a -i pa08a-input0.txt -o pa08a-output0.txt -v
```

Submission

Name your solution **pa08a.cpp** and use the turnin command to submit
your solution.

```
$ turnin –c cs135–kmess –p pa08a –v pa08a.cpp
```