CYBERCAFE EDITORIAL

PROBLEM STATEMENT

Bhavika has decided to open a new cybercafe in her locality but has no idea how many computers will be sufficient. So she decided to start with a few computers and will purchase new ones if needed. She can't maintain a record of people at each moment to find the requirement but she has recorded the visiting and leaving time of each person for the past few days and want to invest according to the past record.
Help her to find if the computers she has are sufficient or she needs more.

Input format

First-line contains n and m the no. of persons visited on a day and no. of computers she already had.
Next line contains visiting time of n persons and second line cntain leaving time of n persons.
0<n,m<100
1<k<10^5

Output format

Print the minimum number of computers she needs to purchase such that no person has to wait.
Note: format of time eg.11:30 will be represented as 1130.

EXPLAINATION

Given visiting and leaving times of all persons that reach cybercafe, the task is to find the minimum number of computers required for the people so that no person waits and then subtract from it the initial no. of computers. We are given two arrays which represent visiting and leaving times of trains that stop.

- **Approach:** The idea is to take every interval one by one and find the number of intervals that overlap with it. Keep track of the maximum number of intervals that overlap with an interval. Finally, return the maximum value.
- **Algorithm:**
    1. Run two nested loops the outer loop from start to end and inner loop from i+1 to end.
    2. For every iteration of outer loop find the count of intervals that intersect with the current interval.
    3. Update the answer with maximum count of overlap in each iteration of outer loop.
    4. Print the answer.

Complexity Analysis:

- **Time Complexity:** O(n^2).
  Two nested loops traverse the array, so the time complexity is O(n^2).
- **Space Complexity:** O(1).
  As no extra space is required.

Efficient Solution:

- **Approach:** The idea is to consider all events in sorted order. Once the events are in sorted order, trace the number of persons at any time keeping track of persons that have arrived, but not departed.
  For example consider the above example.

```
arr[]  = {9:00,  9:40, 9:50,  11:00, 15:00, 18:00}

dep[]  = {9:10, 12:00, 11:20, 11:30, 19:00, 20:00}
```

```
All events are sorted by time.
Total platforms at any time can be obtained by
subtracting total departures from total arrivals
by that time.

 Time      Event Type     Total Platforms Needed
                               at this Time
 9:00       visited                  1
 9:10       left                     0
 9:40       visited                  1
 9:50       visited                  2
 11:00      visited                  3
 11:20      left                     2
 11:30      left                     1
 12:00      left                     0
 15:00      visited                  1
 18:00      visited                  2
 19:00      left                     1
 20:00      left                     0

Minimum computers needed in cybercafe
= Maximum computers needed at any time
= 3
```

- **Algorithm:**
  1. Sort the visiting and leaving time of computers.
  2. Create two pointers i=0, and j=0 and a variable to store *ans* and current count *computers*
  3. Run a loop while i<n and j<n and compare the ith element of visiting array and jth element of leaving array.
  4. if the visiting time is less than or equal to leaving then one more computer is needed so increase the count, i.e. comp++ and increment i
  5. Else if the visiting time greater than leaving then one less platform is needed so decrease the count, i.e. comp++ and increment j
  6. Update the ans, i.e ans = max(ans, comp).

- **Implementation:** This doesn't create a single sorted list of all events, rather it individually sorts arr[] and dep[] arrays, and then uses merge process of merge sort to process them together as a single sorted array.

```
#include<bits/stdc++.h>
using namespace std;
long long int vis[100000];
long long int lev[100000];
long long int calculate(long long int n)
{
    if(n==0)
        return 0;
    sort(vis, vis + n);
    sort(lev, lev + n);

    // comp_needed indicates number of computers
    // needed at a time
    long long int comp_needed = 1, result = 1;
    long long int i = 1, j = 0;

    // Similar to merge in merge sort to process
    // all events in sorted order
    while (i < n && j < n) {
        // If next event in sorted order is arrival,
        // increment count of computers needed
        if (vis[i] <= lev[j]) {
            comp_needed++;
            i++;
        }

        // Else decrement count of computers needed
        else if (vis[i] > lev[j]) {
            comp_needed--;
            j++;
        }

        // Update result if needed
        if (comp_needed > result)
            result = comp_needed;
    }

    return result;
}
int main()
{
    long long int initial_comps,i,j,person_count,max_req=0;
    cin>>person_count>>initial_comps;
    for(i=0;i<person_count;i++)
        cin>>vis[i];
    for(i=0;i<person_count;i++)
        cin>>lev[i];
    max_req = calculate(person_count);
    if(max_req-initial_comps<=0)
       cout<<"0" ;
    else cout<<max_req-initial_comps;
}
```