[B - Queries about less or equal elements](https://vjudge.net/problem/CodeForces-600B" \t "_blank)

You are given two arrays of integers *a* and *b*. For each element of the second array *bj* you should find the number of elements in array *a* that are less than or equal to the value *bj*.

**Input**

The first line contains two integers *n*, *m* (1 ≤ *n*, *m* ≤ 2·105) — the sizes of arrays *a* and *b*.

The second line contains *n* integers — the elements of array *a* ( - 109 ≤ *ai* ≤ 109).

The third line contains *m* integers — the elements of array *b* ( - 109 ≤ *bj* ≤ 109).

**Output**

Print *m* integers, separated by spaces: the *j*-th of which is equal to the number of such elements in array *a* that are less than or equal to the value *bj*.

**Examples**

**Input**

5 4  
1 3 5 7 9  
6 4 2 8

**Output**

3 2 1 4

**Input**

5 5  
1 2 1 2 5  
3 1 4 1 5

**Output**

4 2 4 2 5

**BEWARE**:  This problem has test cases with big inputs.  If you are getting TLE and are confident that there solution is optimized.  Try using FASTIO and re-submit your code. [JAVA FASTIO template is here](https://mentorpick.com/blog/view/610b38da39bcb60e274e701f).

A brute force approach would be, for each element of array b - doing linear search in array a from left to right to count how many elements are less than or equal to b[i].  Here is the pseudo code.

main()  
    int aSize, bSize;  
    Read aSize, bSize;  
  
    int a[aSize];  
    for i = [0, aSize-1]  
        Read a[i];  
  
    int x;  
    **for i = [0, bSize - 1]**  
        Read x;  
        int countLessThanEqualToXInA = 0;  
  
        **for j = [0, aSize - 1]**  
          if ( a[j] <= x )  
            countLessThanEqualToXInA++; // We can put an else case and break as well early if a was sorted  
  
        Print countLessThanEqualToXInA, " ";

Clearly above approach has Time Complexity of aSize (first for loop) + bSize \* aSize (nested for loops) = O(aSize \* bSize)

Since constraints of problem mentioned aSize and bSize can be max 2 \* 105, hence overall iterations would be O(aSize \* bSize) = O(4 \* 1010) which is more than 108 need to get solution ACCEPTED.

BINARY SEARCH BASED APPROACH

We can easily modify the counting problem to search problem.  Let's get the intuition.

If we sort array a first refer example below

a = {2, 5, 8, 9, 11, 15, 19, 21, 35}

Now based on above array, how can we find count of elements less than or equal to X = 20?  Of course we stop our counting as soon as we find 21.  So can we efficiently find 21 using binary search?    Modified problem statement would be for X find index of element such that element is > X.  In above array index of 21 is 7 which means values less than 21 are certainly 7.

So pseudo code will look like

int **findIndexOfValueGreaterThanX**(int a[], int n, int x){  
      
}  
  
main()  
    int aSize, bSize;  
    Read aSize, bSize;  
  
    int a[aSize];  
    for i = [0, aSize-1]  
        Read a[i];  
  
    sort(a, n); // Use built-in to sort the array, to do binary search  
  
    int x;  
    for i = [0, bSize - 1]  
        Read x;  
        int countLessThanEqualToXInA = findIndexOfValueGreaterThanX(a, n, x); // logN time   
        Print countLessThanEqualToXInA, " ";

So, you can quickly code it now?  Figure out how to complete findIndexOfValueGreaterThanX using modified binary search.

**NOTE**:  Search online if your language provides any built in function for binary search or searching a value bigger than given value (C++ has it for sure namely upper\_bound and lower\_bound functions)

Complexity Analysis

Time Complexity of above approach is N (reading array a) + NlogN (sorting array a) + bSize \* log2(aSize) = O(bSize \* log2(aSize))

for max values of aSize and bSize, Operations => O(2 \* 105\* log2(2 \* 105))  <= 2 \* 105 log2(106) ~= 2 \* 105 log2( 220) = 2 \* 105\* 20 =4 \* 106 which will get AC.