Array intersection

Problem Description: In this problem you are given with two random integer arrays, you have to print their intersection. That is, you have to print all the elements that are present in both the given arrays.
For example: If input is: Size1 of arr1=6

```
                        arr1[] =  2 6 8 5 4 3
                   Size2 of arr2= 4
                        arr2[] = 2 3 4 7
```

Output: Elements present in both the arrays:

```
        2
        3
        4
```

How to approach?
Approach 1: The first approach we can think is that we can run 2 loops**.** In the outer loop pick one element of array 1 and then with the help of an inner loop check whether the same element exists in array 2 or not. If the same element is found then, we found an intersection so we print it, and to handle the case of duplicates, we can replace this element with minus infinity.

Pseudo Code for this approach:

```
Function intersection:
  For i = 0 to i less than size1:
      For j = 0 to j less than size2:
          If arr1[i] equal to arr2[j]:
                Print(arr2[j])
                arr2[j]=minus infinity
                break
```

Time Complexity for this approach: Time complexity for this approach is O($size1 * size2$), which is not good, hence, we are moving to the next approach.

Approach 2: A better solution for this problem is to sort both the arrays. Now, we have to find the intersection of 2 sorted arrays:
1) Use two index variables i and j, initialize them as i = 0, j = 0

2) If arr1[i] is smaller than arr2[j] then increment i.
3) If arr1[i] is greater than arr2[j] then increment j.
4) If both are same then print any of them and increment both i and j.

Time Complexity for this approach: Time complexity for sorting both arrays will be O(size1*log(size1) + size2*log(size2)) and then finding the intersection will have a time complexity of O(size1+size2).

- ❑ Let us dry run the code for:

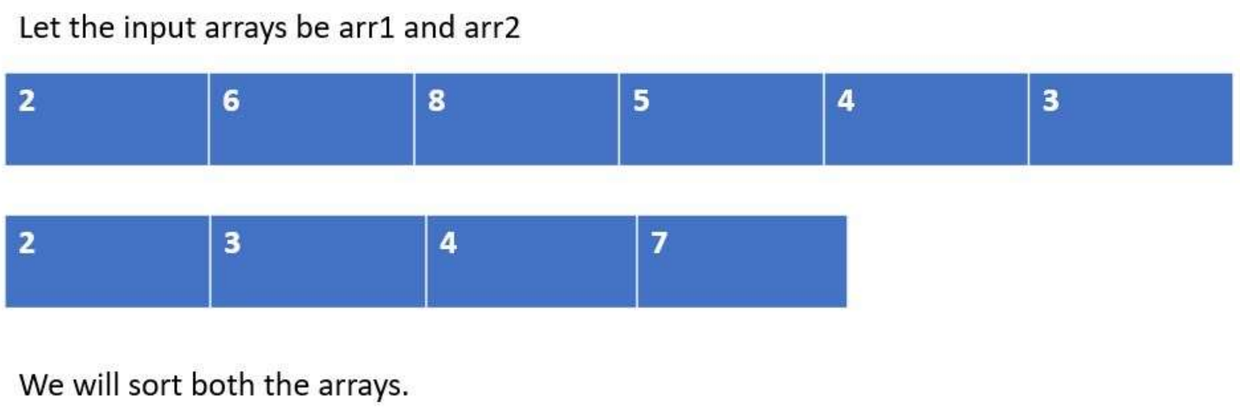
  Size1 of arr1 = 6

  arr1[] = 2 6 8 5 4 3

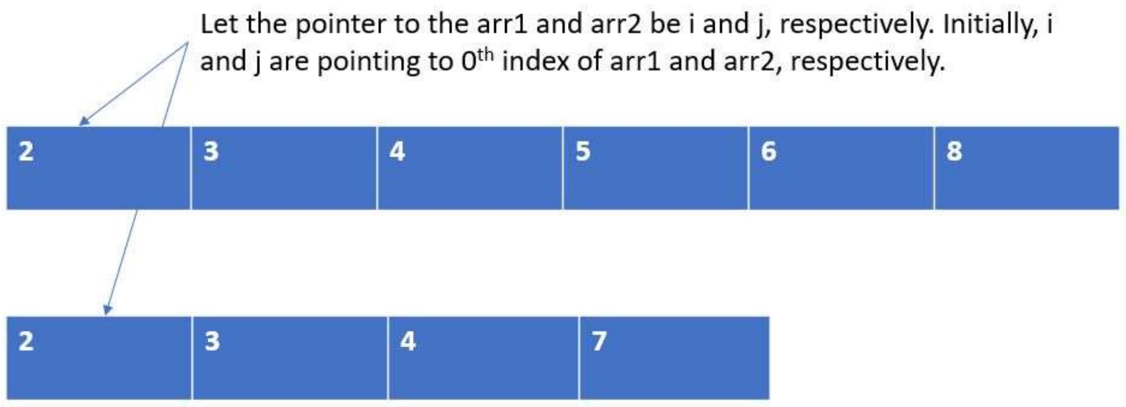
  Size2 of arr2 = 4

  arr2[] = 2 3 4 7

1.

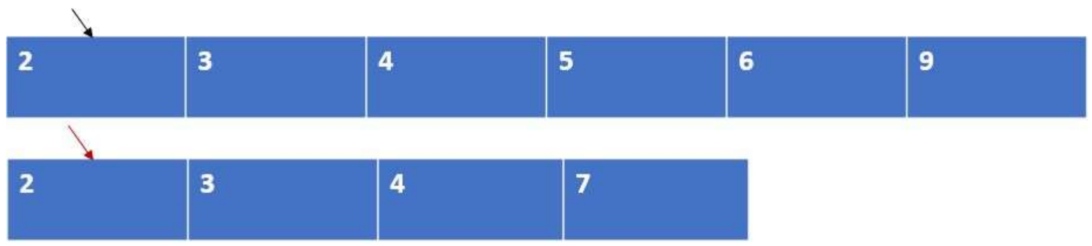

Let the input arrays be arr1 and arr2
2
6
8
5
4
3
2
3
4
7
We will sort both the arrays.

2.


Let the pointer to the arr1 and arr2 be i and j, respectively. Initially, i and j are pointing to 0th index of arr1 and arr2, respectively.
2
3
4
5
6
8
2
3
4
7

3.

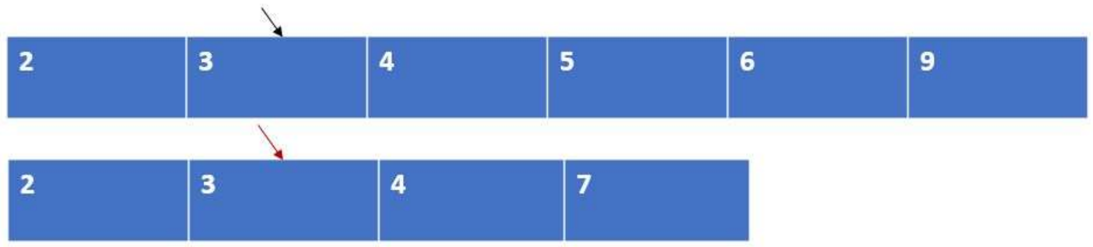
We will compare element at i (represented by black arrow) and j (represented by red arrow). Initially, i and j are pointing to smallest elements of the arrays. We will move the pointer, which is pointing to smaller element because we want to print their intersection and next intersection can only be found by moving pointer pointing to smaller element. In this case, since elements pointed are equal, so, we will print the element in console and move both pointers.



Console:
2

4.

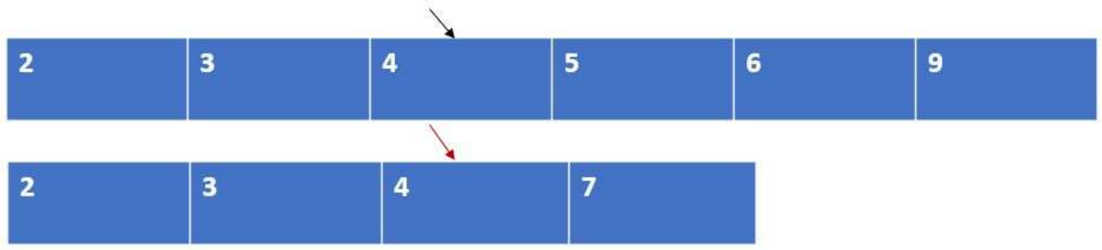
We will again compare element at i and j. In this case, as well, since elements pointed are equal, so, we will print the element in console and move both pointers.



Console:
2 3

5.

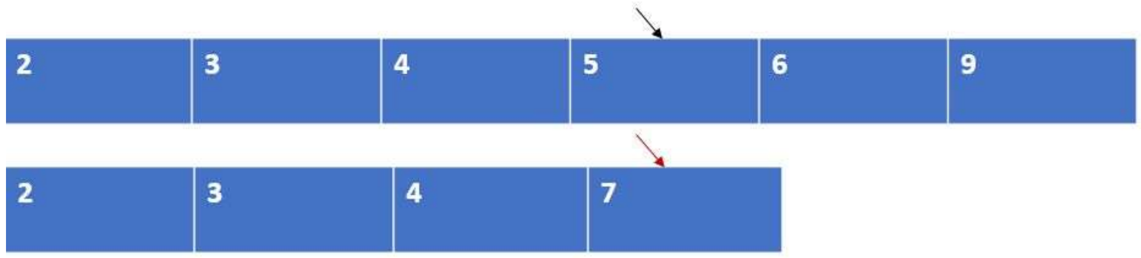
We will again compare element at i and j. In this case, as well, since elements pointed are equal, so, we will print the element in console and move both pointers.



Console:
2 3 4

6.

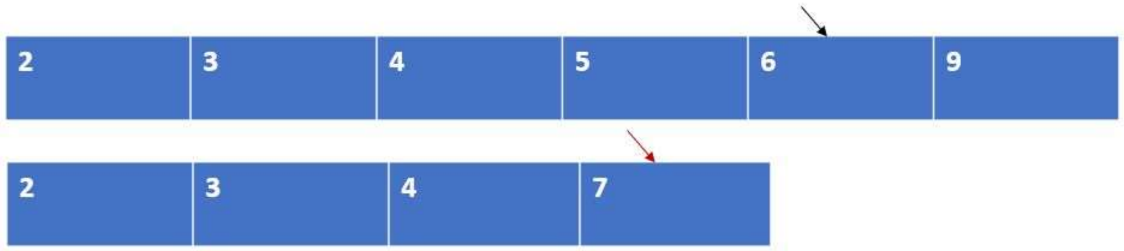
We will again compare element at i and j. In this case, since elements pointed by i is smaller than that pointed by j, so, we will move ith pointer.



Console:
2 3 4

7.

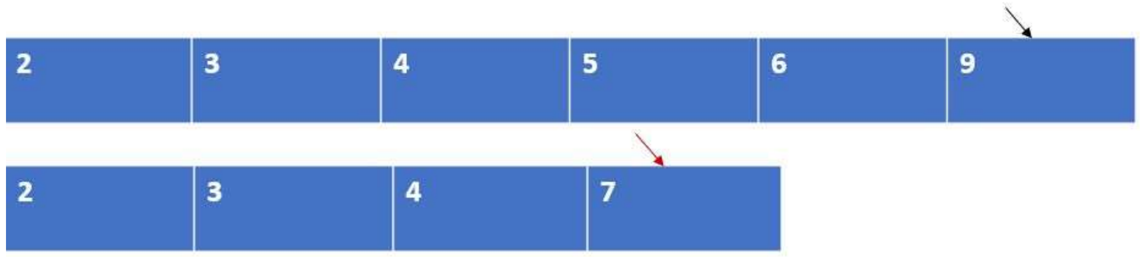
We will again compare element at i and j. In this case, since elements pointed by i is smaller than that pointed by j, so, we will move ith pointer.



Console:
2 3 4

8.

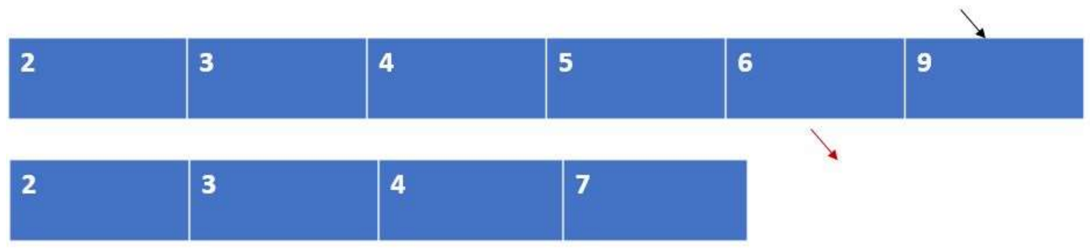
We will again compare element at i and j. In this case, since elements pointed by j is smaller than that pointed by i, so, we will move jth pointer.



Console:
2 3 4

9.

We can’t compare the elements now, as we have reached the end of second array. We can terminate the algorithm now, as no more intersections can be found now.



Console:
2 3 4

Approach 3: The best solution here is to use hashmaps to further reduce the time complexity of this problem. To continue with the hashmaps we can proceed with the following steps:

1. Initialize an empty hashmap mapp.
2. Iterate through the first array, and put every element of this array in the mapp with its corresponding count.
3. Now for every element of the second array, search it in the hashmap and if it is present then print it and decrement its corresponding count. After decrement, if the corresponding count becomes zero, then we should remove the element from the mapp.

Time Complexity for this approach: Time complexity for this approach is O(m+n) as searching and inserting operations in hashmaps are performed in O(1) time.

Pseudo Code for this approach:

```
Function intersection:
    Create an empty hashmap mapp
    For i=0 to i less than size1:
        Increment the count of each element of this array in hashmap
  For i=0 to i less than size2:
       If any element of array2 exists in hashmap:
              Print(element)
             Decrement the count of that element in hashmap.
      If count corresponding to that element is zero
             Delete(element, mapp)
```

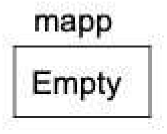
- ❑ Let us dry run the code for:

  Size1 of arr1 = 6

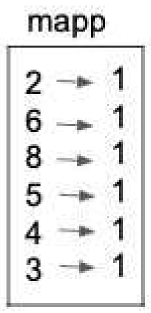
  arr1[] = 2 6 8 5 4 3

  Size2 of arr2 = 4

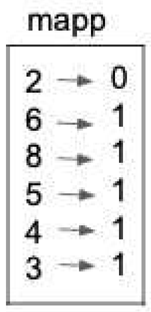
  arr2[] = 2 3 4 7



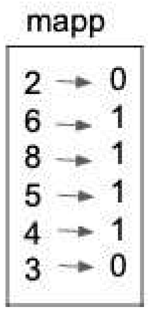
Creating an empty hashmap



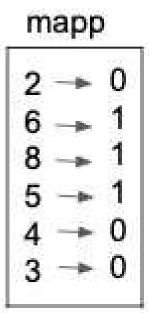
Inserting each element of array1 with its count in the hashmap



Going through each element of array2, here the first element is 2 and since 2 exists in hashmap so, print it and then decrement its count from the mapp and now move to the next element.



Now, the second element 3 exists in hashmap so print it and then decrement its count from the mapp and now move to the next element.



The third element, 4 exists in hashmap so print it and then decrement its count from the mapp and now move to the next element.

Now, when we move to 7, it does not exist in the hashmap, so, we do nothing.

Our final output will be:

2

3

4