How to check if two given sets are disjoint?

Given two sets represented by two arrays, how to check if the given two sets are disjoint or not? It may be assumed that the given arrays have no duplicates.

Difficulty Level: Rookie

```
Input: set1[] = {12, 34, 11, 9, 3}
       set2[] = {2, 1, 3, 5}
Output: Not Disjoint
3 is common in two sets.

Input: set1[] = {12, 34, 11, 9, 3}
       set2[] = {7, 2, 1, 5}
Output: Yes, Disjoint
There is no common element in two sets.
```

We strongly recommend to minimize your browser and try this yourself first.

There are plenty of methods to solve this problem, it's a good test to check how many solutions you can guess.

Method 1 (Simple)

Iterate through every element of first set and search it in other set, if any element is found, return false. If no element is found, return tree. Time complexity of this method is O(mn).

Following is C++ implementation of above idea.

```
// A Simple C++ program to check if two sets are disjoint
#include<iostream>
using namespace std;

// Returns true if set1[] and set2[] are disjoint, else false
bool areDisjoint(int set1[], int set2[], int m, int n)
{
    // Take every element of set1[] and search it in set2
    for (int i=0; i<m; i++)
      for (int j=0; j<n; j++)
         if (set1[i] == set2[j])
            return false;

    // If no element of set1 is present in set2
    return true;
}

// Driver program to test above function
int main()
{
    int set1[] = {12, 34, 11, 9, 3};
    int set2[] = {7, 2, 1, 5};
    int m = sizeof(set1)/sizeof(set1[0]);
    int n = sizeof(set2)/sizeof(set2[0]);
    areDisjoint(set1, set2, m, n)? cout << "Yes" : cout << " No";
    return 0;
}
```

Output:

```
Yes
```

Method 2 (Use Sorting and Merging)

1) Sort first and second sets.

2) Use merge like process to compare elements.

Following is C++ implementation of above idea.

```
// A Simple C++ program to check if two sets are disjoint
#include<iostream>
#include<algorithm>
using namespace std;

// Returns true if set1[] and set2[] are disjoint, else false
bool areDisjoint(int set1[], int set2[], int m, int n)
{
    // Sort the given two sets
    sort(set1, set1+m);
    sort(set2, set2+n);

    // Check for same elements using merge like process
    int i = 0, j = 0;
    while (i < m && j < n)
    {
        if (set1[i] < set2[j])
            i++;
        else if (set2[j] < set1[i])
            j++;
        else /* if set1[i] == set2[j] */
            return false;
    }

    return true;
}

// Driver program to test above function
int main()
{
    int set1[] = {12, 34, 11, 9, 3};
    int set2[] = {7, 2, 1, 5};
    int m = sizeof(set1)/sizeof(set1[0]);
    int n = sizeof(set2)/sizeof(set2[0]);
    areDisjoint(set1, set2, m, n)? cout << "Yes" : cout << " No";
    return 0;
}
```

Output:

```
Yes
```

Time complexity of above solution is O(mLogm + nLogn).

The above solution first sorts both sets, then takes O(m+n) time to find intersection. If we are given that the input sets are sorted, then this method is best among all.

Method 3 (Use Sorting and Binary Search)
This is similar to method 1. Instead of linear search, we use Binary Search.
1) Sort first set.
2) Iterate through every element of second set, and use binary search to search every element in first set. If element is found return it.

Time complexity of this method is O(mLogm + nLogm)

Method 4 (Use Binary Search Tree)
1) Create a self balancing binary search tree (Red Black, AVL, Splay, etc) of all elements in first set.
2) Iterate through all elements of second set and search every element in the above constructed Binary Search Tree. If element is found, return false.
3) If all elements are absent, return true.

Time complexity of this method is O(mLogm + nLogm).

Method 5 (Use Hashing)
1) Create an empty hash table.
2) Iterate through the first set and store every element in hash table.
3) Iterate through second set and check if any element is present in hash table. If present, then return false, else ignore the element.
4) If all elements of second set are not present in hash table, return true.

Following is Java implementation of this method.

```
/* Java program to check if two sets are distinct or not */
import java.util.*;

class Main
{
    // This function prints all distinct elements
    static boolean areDisjoint(int set1[], int set2[])
    {
        // Creates an empty hashset
        HashSet<Integer> set = new HashSet<>();

        // Traverse the first set and store its elements in hash
        for (int i=0; i<set1.length; i++)
            set.add(set1[i]);

        // Traverse the second set and check if any element of it
        // is already in hash or not.
        for (int i=0; i<set2.length; i++)
            if (set.contains(set2[i]))
                return false;

        return true;
    }

    // Driver method to test above method
    public static void main (String[] args)
    {
        int set1[] = {10, 5, 3, 4, 6};
        int set2[] = {8, 7, 9, 3};
        if (areDisjoint(set1, set2)
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

Output:

```
Yes
```

Time complexity of the above implementation is O(m+n) under the assumption that hash set operations like add() and contains() work in O(1) time.