Find Union and Intersection of two unsorted arrays

Given two unsorted arrays that represent two sets (elements in every array are distinct), find union and intersection of two arrays.

For example, if the input arrays are:
arr1[] = {7, 1, 5, 2, 3, 6}
arr2[] = {3, 8, 6, 20, 7}
Then your program should print Union as {1, 2, 3, 5, 6, 7, 8, 20} and Intersection as {3, 6}. Note that the elements of union and intersection can be printed in any order.

Method 1 (Naive)

Union:

1) Initialize union U as empty.
2) Copy all elements of first array to U.
3) Do following for every element x of second array:
.....a) If x is not present in first array, then copy x to U.
4) Return U.

Intersection:

1) Initialize intersection I as empty.
2) Do following for every element x of first array
.....a) If x is present in second array, then copy x to I.
4) Return I.

Time complexity of this method is O(mn) for both operations. Here m and n are number of elements in arr1[] and arr2[] respectively.

Method 2 (Use Sorting)

1) Sort arr1[] and arr2[]. This step takes O(mLogm + nLogn) time.
2) Use O(m + n) algorithms to find union and intersection of two sorted arrays.

Overall time complexity of this method is O(mLogm + nLogn).

Method 3 (Use Sorting and Searching)

Union:

1) Initialize union U as empty.

2) Find smaller of m and n and sort the smaller array.

3) Copy the smaller array to U.

4) For every element x of larger array, do following

…….b) Binary Search x in smaller array. If x is not present, then copy it to U.

5) Return U.

Intersection:

1) Initialize intersection I as empty.

2) Find smaller of m and n and sort the smaller array.

3) For every element x of larger array, do following

…….b) Binary Search x in smaller array. If x is present, then copy it to I.

4) Return I.

Time complexity of this method is min(mLogm + nLogm, mLogn + nLogn) which can also be written as O((m+n)Logm, (m+n)Logn). This approach works much better than the previous approach when difference between sizes of two arrays is significant.

Thanks to use_the_force for suggesting this method in a comment here.

Below is C++ implementation of this method.

```
// A C++ program to print union and intersection of two unsorted arrays
#include <iostream>
#include <algorithm>
using namespace std;

int binarySearch(int arr[], int l, int r, int x);

// Prints union of arr1[0..m-1] and arr2[0..n-1]
void printUnion(int arr1[], int arr2[], int m, int n)
{
    // Before finding union, make sure arr1[0..m-1] is smaller
    if (m > n)
    {
        int *tempp = arr1;
        arr1 = arr2;
        arr2 = tempp;

        int temp = m;
        m = n;
        n = temp;
    }

    // Now arr1[] is smaller

    // Sort the first array and print its elements (these two
    // steps can be swapped as order in output is not important)
    sort(arr1, arr1 + m);
    for (int i=0; i<m; i++)
        cout << arr1[i] << " ";

    // Search every element of bigger array in smaller array
    // and print the element if not found
    for (int i=0; i<n; i++)
        if (binarySearch(arr1, 0, m-1, arr2[i]) == -1)
            cout << arr2[i] << " ";
```

```
}

// Prints intersection of arr1[0..m-1] and arr2[0..n-1]
void printIntersection(int arr1[], int arr2[], int m, int n)
{
    // Before finding intersection, make sure arr1[0..m-1] is smaller
    if (m > n)
    {
        int *tempp = arr1;
        arr1 = arr2;
        arr2 = tempp;

        int temp = m;
        m = n;
        n = temp;
    }

    // Now arr1[] is smaller

    // Sort smaller array arr1[0..m-1]
    sort(arr1, arr1 + m);

    // Search every element of bigger array in smaller array
    // and print the element if found
    for (int i=0; i<n; i++)
        if (binarySearch(arr1, 0, m-1, arr2[i]) != -1)
            cout << arr2[i] << " ";
}

// A recursive binary search function. It returns location of x in
// given array arr[l..r] is present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - l)/2;

        // If the element is present at the middle itself
        if (arr[mid] == x)  return mid;

        // If element is smaller than mid, then it can only be present
        // in left subarray
        if (arr[mid] > x) return binarySearch(arr, l, mid-1, x);

        // Else the element can only be present in right subarray
        return binarySearch(arr, mid+1, r, x);
    }

    // We reach here when element is not present in array
    return -1;
}

/* Driver program to test above function */
int main()
{
    int arr1[] = {7, 1, 5, 2, 3, 6};
    int arr2[] = {3, 8, 6, 20, 7};
    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);
    cout << "Union of two arrays is \n";
    printUnion(arr1, arr2, m, n);
    cout << "\nIntersection of two arrays is \n";
    printIntersection(arr1, arr2, m, n);
    return 0;
}
```

Run on IDE

Output:

```
Union of two arrays is
3 6 7 8 20 1 5 2
Intersection of two arrays is
7 3 6
```

Method 4 (Use Hashing)

Union:

1) Initialize union U as empty.
1) Initialize an empty hash table.
2) Iterate through first array and put every element of first array in the hash table, and in U.
4) For every element x of second array, do following
…….a) Search x in the hash table. If x is not present, then copy it to U.
5) Return U.

Intersection:

1) Initialize intersection I as empty.
2) In initialize an empty hash table.
3) Iterate through first array and put every element of first array in the hash table.
4) For every element x of second array, do following
…….a) Search x in the hash table. If x is present, then copy it to I.
5) Return I.

Time complexity of this method is Θ(m+n) under the assumption that hash table search and insert operations take Θ(1) time.

Please write comments if you find any bug in above codes/algorithms, or find other ways to solve the same problem.

32 Comments Category: Arrays

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average
- Count triplets with sum smaller than a given value
- Find zeroes to be flipped so that number of consecutive 1's is maximized
- Reorder an array according to given indexes
- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- Find maximum average subarray of k length

(Login to Rate and Mark)

3 Average Difficulty : **3/5.0**
Based on **8** vote(s)

Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.