Find four elements a, b, c and d in an array such that a+b = c+d

Given an array of distinct integers, find if there are two pairs (a, b) and (c, d) such that a+b = c+d, and a, b, c and d are distinct elements. If there are multiple answers, then print any of them.

Example:

```
Input:   {3, 4, 7, 1, 2, 9, 8}
Output:  (3, 8) and (4, 7)
Explanation: 3+8 = 4+7

Input:   {3, 4, 7, 1, 12, 9};
Output:  (4, 12) and (7, 9)
Explanation: 4+12 = 7+9

Input:  {65, 30, 7, 90, 1, 9, 8};
Output:  No pairs found
```

Expected Time Complexity: O(n^2)

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

A **Simple Solution** is to run four loops to generate all possible quadruples of array element. For every quadruple (a, b, c, d), check if (a+b) = (c+d). Time complexity of this solution is O(n^4).

An **Efficient Solution** can solve this problem in O(n^2) time. The idea is to use hashing. We use sum as key and pair as value in hash table.

```
Loop i = 0 to n-1 :
    Loop j = i + 1 to n-1 :
        calculate sum
        If in hash table any index already exist
            Then print (i, j) and previous pair
            from hash table
        Else update hash table
    EndLoop;
EndLoop;
```

Below are implementations of above idea. In below implementation, map is used instead of hash. Time complexity of map insert and search is actually O(Log n) instead of O(1). So below implementation is O(n^2 Log n).

C/C++

```cpp
// Find four different elements a,b,c and d of array such that
// a+b = c+d
#include<bits/stdc++.h>
using namespace std;

bool findPairs(int arr[], int n)
{
    // Create an empty Hash to store mapping from sum to
    // pair indexes
    map<int, pair<int, int> > Hash;

    // Traverse through all possible pairs of arr[]
    for (int i = 0; i < n; ++i)
    {
        for (int j = i + 1; j < n; ++j)
        {
            // If sum of current pair is not in hash,
            // then store it and continue to next pair
            int sum = arr[i] + arr[j];
            if (Hash.find(sum) == Hash.end())
                Hash[sum] = make_pair(i, j);

            else // Else (Sum already present in hash)
            {
                // Find previous pair
                pair<int, int> pp = Hash[sum];// pp->previous pair

                // Since array elements are distinct, we don't
                // need to check if any element is common among pairs
                cout << "(" << arr[pp.first] << ", " << arr[pp.second]
                     << ") and (" << arr[i] << ", " << arr[j] << ")\n";
                return true;
            }
        }
    }

    cout << "No pairs found";
    return false;
}

// Driver program
int main()
{
    int arr[] = {3, 4, 7, 1, 2, 9, 8};
    int n  =  sizeof arr / sizeof arr[0];
    findPairs(arr, n);
    return 0;
}
```

Java

```java
// Java Program to find four different elements a,b,c and d of
// array such that a+b = c+d
import java.io.*;
import java.util.*;

class ArrayElements
{
    // Class to represent a pair
    class pair
    {
        int first, second;
        pair(int f,int s)
        {
            first = f; second = s;
        }
    };

    boolean findPairs(int arr[])
    {
        // Create an empty Hash to store mapping from sum to
        // pair indexes
        HashMap<Integer,pair> map = new HashMap<Integer,pair>();
        int n=arr.length;

        // Traverse through all possible pairs of arr[]
        for (int i=0; i<n; ++i)
        {
            for (int j=i+1; j<n; ++j)
            {
                // If sum of current pair is not in hash,
                // then store it and continue to next pair
                int sum = arr[i]+arr[j];
                if (!map.containsKey(sum))
                    map.put(sum,new pair(i,j));

                else // Else (Sum already present in hash)
                {
                    // Find previous pair
                    pair p = map.get(sum);

                    // Since array elements are distinct, we don't
                    // need to check if any element is common among pairs
                    System.out.println("("+arr[p.first]+", "+arr[p.second]+
                                  ") and ("+arr[i]+", "+arr[j]+")");
                    return true;
                }
            }
        }
        return false;
    }

    // Testing program
    public static void main(String args[])
    {
        int arr[] = {3, 4, 7, 1, 2, 9, 8};
        ArrayElements a = new ArrayElements();
        a.findPairs(arr);
    }
}
// This code is contributed by Aakash Hasija
```

Output:

```
(3, 8) and (4, 7)
```

Thanks to Gaurav Ahirwar for suggesting above solutions.

Exercise:

1) Extend the above solution with duplicates allowed in array.

2) Further extend the solution to print all quadruples in output instead of just one. And all quadruples should be printed printed in

lexicographical order (smaller values before greater ones). Assume we have two solutions S1 and S2.

```
S1 : a1 b1 c1 d1 ( these are values of indices int the array )
S2 : a2 b2 c2 d2

S1 is lexicographically smaller than S2 iff
  a1 < a2 OR
  a1 = a2 AND b1 < b2 OR
  a1 = a2 AND b1 = b2 AND c1 < c2 OR
  a1 = a2 AND b1 = b2 AND c1 = c2 AND d1 < d2
```

See this for solution of exercise.