Check if an array can be divided into pairs whose sum is divisible by k

Given an array of integers and a number k, write a function that returns true if given array can be divided into pairs such that sum of every pair is divisible by k.

Examples:

```
Input: arr[] = {9, 7, 5, 3}, k = 6
Output: True
We can divide array into (9, 3) and (7, 5).
Sum of both of these pairs is a multiple of 6.

Input: arr[] = {92, 75, 65, 48, 45, 35}, k = 10
Output: True
We can divide array into (92, 48), (75, 65) and
(45, 35). Sum of all these pairs is a multiple of 10.

Input: arr[] = {91, 74, 66, 48}, k = 10
Output: False
```

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

A **Simple Solution** is to iterate through every element arr[i]. Find if there is another not yet visited element that has remainder as (k – arr[i]%k). If there is no such element, return false. If a pair is found, then mark both elements as visited. Time complexity of this solution is O(n^2 and it requires O(n) extra space.

An **Efficient Solution** is to use Hashing.

```
1) If length of  given array is odd, return false. An odd
   length array cannot be divided in pairs.
2) Traverse input array array and count occurrences of
   all remainders.
      freq[arr[i] % k]++
3) Traverse input array again.
   a) Find remainder of current element.
   b) If remainder divides k into two halves, then
      there must be even occurrences of it as it forms
      pair with itself only.
   c) If remainder is 0, then then there must be even
      occurrences.
   c) Else, number of occurrences of current remainder
      must be equal to number of occurrences of "k -
      current remainder".
```

Time complexity of above algorithm is O(n).

Below implementation uses map in C++ STL. The map is typically implemented using Red-Black Tree and takes O(Log n) time for access. Therefore time complexity of below implementation is O(n Log n), but the algorithm can be easily implemented in O(n) time using hash table.

```cpp
// A C++ program to check if arr[0..n-1] can be divided
// in pairs such that every pair is divisible by k.
#include <bits/stdc++.h>
using namespace std;

// Returns true if arr[0..n-1] can be divided into pairs
// with sum divisible by k.
bool canPairs(int arr[], int n, int k)
{
    // An odd length array cannot be divided into pairs
    if (n & 1)
        return false;

    // Create a frequency array to count occurrences
    // of all remainders when divided by k.
    map<int, int> freq;

    // Count occurrences of all remainders
    for (int i = 0; i < n; i++)
        freq[arr[i] % k]++;

    // Traverse input array and use freq[] to decide
    // if given array can be divided in pairs
    for (int i = 0; i < n; i++)
    {
        // Remainder of current element
        int rem = arr[i] % k;

        // If remainder with current element divides
        // k into two halves.
        if  (2*rem == k)
        {
            // Then there must be even occurrences of
            // such remainder
            if (freq[rem] % 2 != 0)
                return false;
        }

        // If remainder is 0, then there must be two
        // elements with 0 remainder
        else if (rem == 0)
        {
           if (freq[rem] & 1)
               return false;
        }

        // Else number of occurrences of remainder
        // must be equal to number of occurrences of
        // k - remainder
        else if (freq[rem] != freq[k - rem])
            return false;
    }
    return true;
}

/* Driver program to test above function */
int main()
{
    int arr[] =  {92, 75, 65, 48, 45, 35};
    int k = 10;
    int n = sizeof(arr)/sizeof(arr[0]);
    canPairs(arr, n, k)? cout << "True": cout << "False";
    return 0;
}
```

Output:

```
True
```