# Count Number Pairs that Add Up to K

September 25, 2017

## Question

Given an array of integers, and a number $k$, find the number of pairs of integers in the array whose sum is equal to $k$.

## Explanation and Algorithm

This is a unique problem because the first way you'd think of doing it is with a nested for-loop through the array to count the pairs. This is a solution, but is it the best one? No, because you could actually use a hash table to find this.

## Hints

1. First think of the brute force method. It's not expected that you'll arrive at the optimal solution.
2. Now, think of how you can utilize an Integer Hashmap to find the sums and keep track of the count. Remember the different Hashmap methods like containsKey, put, search, and get. These will all be useful.
3. How should we use these methods when ints adding up to k are found? Otherwise? Will the number we increment be the actual count or do we need to multiply it by something in order to get to that count?
4. Think of how you'd deal with repeat-numbers or other edge cases.

## Code

---

```
/*Answer 1: Brute Force*/

public int pairs(int [] a, int k){
```

```
    int count = 0;
   for(int i = 0; i< a.length){
      for(int j = i+1; j<n; j++){
         if(a[i]+a[j] == k){
               count++;
            }

         }

    }
   return count;
}
```

---

```
/*Answer 2: Hash Table Approach */

public int pairs(int[]a, int k){

HashMap<Integer, Integer> search = new HashMap<>();

   for(int i = 0; i < a.length; i++){
      if(!search.containsKey(a[i])){
         search.put(a[i],0);
       }
       search.put(a[i], search.get(a[i+1]+1);
      }

      int twiceCount = 0;

        for(int i = 0; i < a.length; i++){
         if(search.get(k-a[i]) != null){
            twiceCount += search.get(k-a[i]);
            }

            if(k-a[i] == a[i]){
               twiceCount--;
            }
        }
        return twiceCount/2;
}
```

---

## Run time analysis

For the brute force solution, the running time is $O(n^2)$ because in each for-loop you are traversing the array $n$ times. Therefore, the array is being traversed a total of $n^2$ times.

For the hash table solution, the running time is $O(n)$ because we only need one traversal of the array. This means we go through the array $n$ times.

## Sources

Found on HackerRank Challenge. **This was an interview question for me**.