Distribute Candies

Given an integer array with **even** length, where different numbers in this array represent different **kinds** of candies. Each number means one candy of the corresponding kind. You need to distribute these candies **equally** in number to brother and sister. Return the maximum number of **kinds** of candies the sister could gain.

Example 1:

```
Input: candies = [1,1,2,2,3,3]
Output: 3
Explanation:
There are three different kinds of candies (1, 2 and 3), and two candies for each kind.
Optimal distribution: The sister has candies [1,2,3] and the brother has candies [1,2,3], too.
The sister has three different kinds of candies.
```

Example 2:

```
Input: candies = [1,1,2,3]
Output: 2
Explanation: For example, the sister has candies [2,3] and the brother has candies [1,1].
The sister has two different kinds of candies, the brother has only one kind of candies.
```

Note:

1. The length of the given array is in range [2, 10,000], and will be even.
2. The number in given array is in range [-100,000, 100,000].

Solution 1

Thanks @wmcalyj , modified to use HashSet.

```
public class Solution {
    public int distributeCandies(int[] candies) {
        Set<Integer> kinds = new HashSet<>();
        for (int candy : candies) kinds.add(candy);
        return kinds.size() >= candies.length / 2 ? candies.length / 2 : kinds.size
();
    }
}
```

written by shawngao original link here

Solution 2

There are `len(set(candies))` unique candies, and the sister picks only `len(candies) / 2` of them, so she can't have more than this amount.

For example, if there are 5 unique candies, then if she is picking 4 candies, she will take 4 unique ones. If she is picking 7 candies, then she will only take 5 unique ones.

```
def distributeCandies(self, candies):
    return min(len(candies) / 2, len(set(candies)))
```

written by awice original link here

Solution 3

Set - O(N) time, O(N) space

We can use a set to count all unique kinds of candies, but even all candies are unique, the sister cannot get more than half.
(Even though in reality my GF would always get more than half.)

```
class Solution {
public:
    int distributeCandies(vector<int>& candies) {
        unordered_set<int> kinds;
        for (int kind : candies) {
            kinds.insert(kind);
        }
        return min(kinds.size(), candies.size() / 2);
    }
};
```

Using range constructor as @i_square suggested:

```
class Solution {
public:
    int distributeCandies(vector<int>& candies) {
        return min(unordered_set<int>(candies.begin(), candies.end()).size(), candi
es.size() / 2);
    }
};
```

Sort - O(N logN) time, O(1) space

Or we can sort the candies by kinds, count kind if different than the prevous:

```
class Solution {
public:
    int distributeCandies(vector<int>& candies) {
        size_t kinds = 0;
        sort(candies.begin(), candies.end());
        for (int i = 0; i < candies.size(); i++) {
            kinds += i == 0 || candies[i] != candies[i - 1];
        }
        return min(kinds, candies.size() / 2);
    }
};
```

The counter can start from 1 since there is no test case for empty input:

```cpp
class Solution {
public:
    int distributeCandies(vector<int>& candies) {
        size_t kinds = 1;
        sort(candies.begin(), candies.end());
        for (int i = 0; i < candies.size(); i++) {
            kinds += candies[i] != candies[i - 1];
        }
        return min(kinds, candies.size() / 2);
    }
};
```

written by alexander original link here