JavaScript Distribute Candies

Challenge

Trev has `n` candies, where the `i^th` candy is of type `candyType[i]`. Trev noticed that he started to gain weight, so he visited a doctor.

The doctor advised Trev to only eat `n / 2` of the candies he has (`n` is always even). Trev likes his candies very much, and he wants to eat the maximum number of different types of candies while still following the doctor's advice.

Given the integer array `candyType` of length `n`, return the maximum number of different types of candies he can eat if he only eats `n / 2` of them.

1st Example

```
Input: candyType = [1,1,2,2,3,3]
Output: 3
Explanation: Trev can only eat 6 / 2 = 3 candies. Since
             there are only 3 types, he can eat one of each
             type.
```

2nd Example

```
Input: candyType = [1,1,2,3]
Output: 2
Explanation: Trev can only eat 4 / 2 = 2 candies. Whether
             he eats types [1,2], [1,3], or [2,3], he still
             can only eat 2 different types.
```

3rd Example

```
Input: candyType = [6,6,6,6]
Output: 1
Explanation: Trev can only eat 4 / 2 = 2 candies. Even
             though he can eat 2 candies, he only has 1 type.
```

Constraints

- `n == candyType.length`
- `2 <= n <= 10⁴`
- `-10⁵ <= candyType[i] <= 10⁵`
- `n` is even.

Solution

```
const distributeCandies = (candyType) => {
    const hashMap = {};

    let output = 0;
```

Solution continues on next page...

```
    for (let i = 0; i < candyType.length; i++) {
        if (!hashMap[candyType[i]]) {
            output++;
        }

        hashMap[candyType[i]] = i + 1;

        if (output === Math.floor(candyType.length / 2)) {
            return output;
        }
    }

    return output;
};
```

Explanation

I've created a function called `distributeCandies` that takes an array called `candyType` as its parameter. This function calculates the maximum number of different types of candies that can be distributed to a person by following a certain rule.

The function starts by initializing an empty object called `hashMap` and a variable called `output` with a value of `0`.

It then enters a `for` loop that iterates through each element in the `candyType` array.

Inside the loop, it checks if the current candy type `(candyType[i]` is not already present as a key in the `hashMap` object. If it is not, it means that this is a new type of candy, so the `output` variable is incremented by `1`.

After that, it assigns the value of `i + 1` to the `candyType[i]` key in the `hashMap` object. This is done to keep track of the index of the last occurrence of each candy type.

Next, it checks if the `output` variable is equal to half of the length of the `candyType` array. If it is, it means that the maximum number of different candies that can be distributed has been reached, so the function returns the current value of `output`.

If the condition mentioned above is not met, the loop continues to the next iteration.

After the loop finishes, the function returns the final value of `output`, which represents the maximum number of different candies that can be distributed.

In summary, this function calculates the maximum number of different candies that can be distributed to a person by iterating through an array of candy types and keeping track of the number of unique candy types encountered.

Author: Trevor Morin