<https://leetcode.com/problems/number-of-distinct-averages/description/>

You are given a **0-indexed** integer array nums of **even** length.

As long as nums is **not** empty, you must repetitively:

Find the minimum number in nums and remove it.

Find the maximum number in nums and remove it.

Calculate the average of the two removed numbers.

The **average** of two numbers a and b is (a + b) / 2.

For example, the average of 2 and 3 is (2 + 3) / 2 = 2.5.

Return**the number of distinct averages calculated using the above process**.

**Note** that when there is a tie for a minimum or maximum number, any can be removed.

**Example 1:**

**Input:** nums = [4,1,4,0,3,5]

**Output:** 2

**Explanation:**

1. Remove 0 and 5, and the average is (0 + 5) / 2 = 2.5. Now, nums = [4,1,4,3].

2. Remove 1 and 4. The average is (1 + 4) / 2 = 2.5, and nums = [4,3].

3. Remove 3 and 4, and the average is (3 + 4) / 2 = 3.5.

Since there are 2 distinct numbers among 2.5, 2.5, and 3.5, we return 2.

**Example 2:**

**Input:** nums = [1,100]

**Output:** 1

**Explanation:**There is only one average to be calculated after removing 1 and 100, so we return 1.

**Constraints:**

2 <= nums.length <= 100

nums.length is even.

0 <= nums[i] <= 100

**Attempt 1: 2024-01-27**

**Solution 1: Hash Table (10 min)**

**Style 1: Hash Set**

class Solution {

    public int distinctAverages(int[] nums) {

        Arrays.sort(nums);

        int count = 0;

        Set<Double> set = new HashSet<>();

        int i = 0;

        int j = nums.length - 1;

        while(i < j) {

            set.add((nums[i] + nums[j]) / 2.0);

            i++;

            j--;

        }

        return set.size();

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Style 2: Frequency Array**

class Solution {

    public int distinctAverages(int[] nums) {

        Arrays.sort(nums);

        int count = 0;

        int[] buckets = new int[201];

        int i = 0;

        int j = nums.length - 1;

        while(i < j) {

            buckets[nums[i] + nums[j]]++;

            i++;

            j--;

        }

        for(int k = 0; k < 201; k++) {

            if(buckets[k] > 0) {

                count++;

            }

        }

        return count;

    }

}

Time Complexity: O(N)

Space Complexity: O(N)