Given an integer array nums, return *the number of reverse pairs in the array*.

A reverse pair is a pair (i, j) where:

- **0 <= i < j < nums.length and**
- **nums[i] > 2 * nums[j].**

Example 1:

Input: nums = [1,3,2,3,1]
Output: 2
Explanation: The reverse pairs are:
(1, 4) --> nums[1] = 3, nums[4] = 1, 3 > 2 * 1
(3, 4) --> nums[3] = 3, nums[4] = 1, 3 > 2 * 1

Example 2:

Input: nums = [2,4,3,5,1]
Output: 3
Explanation: The reverse pairs are:
(1, 4) --> nums[1] = 4, nums[4] = 1, 4 > 2 * 1
(2, 4) --> nums[2] = 3, nums[4] = 1, 3 > 2 * 1
(3, 4) --> nums[3] = 5, nums[4] = 1, 5 > 2 * 1

Constraints:

- **1 <= nums.length <= $5 * 10^4$**
- **-2^{31} <= nums[i] <= $2^{31} - 1$**

Solution:

```
class Solution {
   int merge(int[] nums, int low, int mid, int high) {
      int count = 0;
      int j = mid + 1;
      for (int i = low; i <= mid; i++) {
         while (j <= high && nums[i] > (2L * nums[j])) {
            j++;
         }
         count += j - (mid + 1);
      }

      int[] temp = new int[high - low + 1];
      int left = low, right = mid + 1, k = 0;
```

```
        while (left <= mid && right <= high) {
            if (nums[left] <= nums[right]) {
                temp[k++] = nums[left++];
            } else {
                temp[k++] = nums[right++];
            }
        }

        while (left <= mid) {
            temp[k++] = nums[left++];
        }
        while (right <= high) {
            temp[k++] = nums[right++];
        }

        System.arraycopy(temp, 0, nums, low, temp.length);
        return count;
    }

    public int mergeSort(int[] nums, int low, int high) {
        if (low >= high) {
            return 0;
        }
        int mid = (low + high) / 2;
        int inv = mergeSort(nums, low, mid);
        inv += mergeSort(nums, mid + 1, high);
        inv += merge(nums, low, mid, high);
        return inv;
    }

    public int reversePairs(int[] nums) {
        return mergeSort(nums, 0, nums.length - 1);
    }
}
```