▇ Reverse Pairs – Full Documentation

1. Problem Statement

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

A reverse pair is a pair (i, j) such that:

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

2. Intuition

This problem is a variation of counting inversions, which can be efficiently solved using a modified merge sort algorithm. The core idea is to:

- Divide the array into smaller parts,
- Count valid reverse pairs across the two halves,
- Then merge them back in sorted order.

3. Key Observations

- A brute-force $O(n^2)$ approach would time out for large inputs (up to 50,000).
- During merge sort, for each element in the left half, we can count how many elements in the right half satisfy the condition nums[i] > 2 * nums[j].
- Sorting the two halves enables efficient searching and merging.

4. Approach

- Divide and Conquer using Merge Sort:
    - Recursively split the array.
    - Count reverse pairs across subarrays.

    - Use two pointers to count valid pairs before merging.
- Merge Step:
    - Merge two sorted halves.
    - Maintain order to allow efficient comparison.

5. **Edge Cases**

- Array with all elements the same (e.g., [2,2,2]) → No reverse pairs.
- Array sorted in descending order → Maximum reverse pairs.
- Array of length 1 → No pairs.
- Large integers or negatives → Ensure 2 * nums[j] doesn't overflow.

6. **Complexity Analysis**

□ Time Complexity:

- O(n log n), where n is the length of the array.
- Each merge sort step divides the array and counts in linear time.

□ Space Complexity:

- O(n), for the temporary array used in merging.

7. **Alternative Approaches**

- Binary Indexed Tree / Segment Tree:
    - Can be used with coordinate compression.
    - More complex and usually less efficient for this particular problem.
- Brute-force $O(n^2)$:
    - Compare all pairs; not efficient for large input.

8. Test Cases

Input	Output	Explanation
[1,3,2,3,1]	2	Pairs: (1,4) and (3,4)
[2,4,3,5,1]	3	Pairs: (1,4), (2,4), and (3,4)
[5,4,3,2,1]	4	Pairs: (0,3), (0,4), (1,4), (2,4)
[1,1,1,1,1]	0	No pair satisfies nums[i] > 2 * nums[j]
[1]	0	Single element → no pairs

9. Final Thoughts

- This problem teaches efficient problem-solving using modified merge sort.
- It highlights how sorting can help solve complex comparison-based problems.
- Understanding this solution builds a foundation for tackling other inversion-counting problems.