is it correct ? -

class Solution {

public:

long long perfectPairs(vector<int>& nums) {

int count = 0;

if(nums.empty()){

return count;

}

int a = 0;

int b = 0;

for(int i = 0;i<nums.size();i++){

for(int j = i+1;j<nums.size();j++){

a = nums[i];

b = nums[j];

if((min(abs(a-b),abs(a+b))<= min(abs(a),abs(b))) && (max(abs(a),abs(b))<= max(abs(a-b),abs(a+b)))){

count++;

}

}

}

return count;

}

};

You are given an integer array nums.

A pair of indices (i, j) is called perfect if the following conditions are satisfied:

i < j

Let a = nums[i], b = nums[j]. Then:

min(|a - b|, |a + b|) <= min(|a|, |b|)

max(|a - b|, |a + b|) >= max(|a|, |b|)

Return the number of distinct perfect pairs.

Note: The absolute value |x| refers to the non-negative value of x.

Example 1:

Input: nums = [0,1,2,3]

Output: 2

Explanation:

There are 2 perfect pairs:

(i, j) (a, b) min(|a − b|, |a + b|) min(|a|, |b|) max(|a − b|, |a + b|) max(|a|, |b|)

(1, 2) (1, 2) min(|1 − 2|, |1 + 2|) = 1 1 max(|1 − 2|, |1 + 2|) = 3 2

(2, 3) (2, 3) min(|2 − 3|, |2 + 3|) = 1 2 max(|2 − 3|, |2 + 3|) = 5 3

Example 2:

Input: nums = [-3,2,-1,4]

Output: 4

Explanation:

There are 4 perfect pairs:

(i, j) (a, b) min(|a − b|, |a + b|) min(|a|, |b|) max(|a − b|, |a + b|) max(|a|, |b|)

(0, 1) (-3, 2) min(|-3 - 2|, |-3 + 2|) = 1 2 max(|-3 - 2|, |-3 + 2|) = 5 3

(0, 3) (-3, 4) min(|-3 - 4|, |-3 + 4|) = 1 3 max(|-3 - 4|, |-3 + 4|) = 7 4

(1, 2) (2, -1) min(|2 - (-1)|, |2 + (-1)|) = 1 1 max(|2 - (-1)|, |2 + (-1)|) = 3 2

(1, 3) (2, 4) min(|2 - 4|, |2 + 4|) = 2 2 max(|2 - 4|, |2 + 4|) = 6 4

Example 3:

Input: nums = [1,10,100,1000]

Output: 0

Explanation:

There are no perfect pairs. Thus, the answer is 0.

Constraints:

2 <= nums.length <= 105

-109 <= nums[i] <= 109

Note: Please do not copy the description during the contest to maintain the integrity of your submissions.  
  
response -   
Your code \*\*is logically correct\*\* for small inputs, but it has a \*\*major performance issue\*\* that makes it unsuitable for large inputs due to the constraints:

> `2 <= nums.length <= 10^5`

This means your current nested loop approach runs in \*\*O(n²)\*\* time, which will lead to a \*\*Time Limit Exceeded (TLE)\*\* error for large test cases.

---

### ✅ Let's analyze what your code does:

You're checking all pairs `(i, j)` where `i < j`, and verifying whether this pair satisfies the two conditions:

1. `min(|a - b|, |a + b|) <= min(|a|, |b|)`

2. `max(|a - b|, |a + b|) >= max(|a|, |b|)`

Which is exactly what the problem asks for.

So \*\*logically\*\*, yes — your implementation is correct.

But...

---

## ❌ Why it won't work in practice (for big inputs):

- The constraints allow up to \*\*100,000 elements\*\*.

- A brute-force solution with nested loops would perform up to \*\*~5 × 10⁹ operations\*\*, which is way too slow.

---

## 🧠 How to optimize?

We need to find a better way than checking every pair. Let’s look at the mathematical properties of the condition.

### Key Insight:

Let’s denote:

- `x = abs(a)`

- `y = abs(b)`

- Without loss of generality, assume `x <= y`.

Then:

- `min\_val = min(|a - b|, |a + b|)`

- `max\_val = max(|a - b|, |a + b|)`

We want:

- `min\_val <= x`

- `max\_val >= y`

But since `min\_val <= max\_val`, and we're comparing with `x` and `y`, we can derive that:

#### Case Analysis Based on Signs of a and b:

Let’s think about when these conditions are satisfied by considering the signs of `a` and `b`.

There are four combinations:

1. Both positive: `a >= 0, b >= 0`

2. One positive, one negative: `a >= 0, b < 0` or vice versa

3. Both negative: `a < 0, b < 0`

In each case, `|a ± b|` simplifies differently.

Instead of brute-force, we can \*\*sort the array and use binary search or two pointers\*\*, depending on how the values behave under the conditions.

---

## 🔁 Optimized Approach Idea:

Use sorting + case-based analysis + two-pointer technique.

Here's an outline:

1. Separate numbers into groups based on sign.

2. For each group, sort them by absolute value.

3. Use two pointers to efficiently count valid pairs satisfying the inequality.

This brings the time complexity down to \*\*O(n log n)\*\*.

---

## ✅ Corrected Efficient Version (C++):

```cpp

class Solution {

public:

long long perfectPairs(vector<int>& nums) {

vector<int> pos, neg, zero;

for (int x : nums) {

if (x > 0) pos.push\_back(x);

else if (x < 0) neg.push\_back(-x); // store as positive

else zero.push\_back(x);

}

sort(pos.begin(), pos.end());

sort(neg.begin(), neg.end());

long long res = 0;

// Count pairs from same-sign groups

res += countPairs(pos, pos);

res += countPairs(neg, neg);

// Count cross-sign pairs

res += countPairs(pos, neg);

res += countPairs(neg, pos);

// Count zero pairs

long long sz = zero.size();

res += sz \* (sz - 1) / 2;

res += sz \* (nums.size() - sz); // zeros paired with non-zeros

return res;

}

private:

long long countPairs(const vector<int>& a, const vector<int>& b) {

long long cnt = 0;

int i = 0, j = 0;

int n = a.size(), m = b.size();

while (i < n && j < m) {

int x = a[i], y = b[j];

int min\_val = min(abs(x - y), abs(x + y));

int max\_val = max(abs(x - y), abs(x + y));

int min\_xy = min(x, y);

int max\_xy = max(x, y);

if (min\_val <= min\_xy && max\_val >= max\_xy) {

cnt += (m - j);

i++;

} else {

j++;

}

}

return cnt;

}

};

```

---

## ✅ Summary:

| Aspect | Verdict |

|-------|---------|

| Logic Correctness | ✅ Yes |

| Time Complexity | ❌ O(n²) – too slow for large input |

| Space Complexity | ✅ O(1) |

| Recommended Fix | ✅ Sort + two-pointer approach |

Let me know if you'd like a step-by-step explanation of the optimized version!