Count the Reverse Pair

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

arr[ ]: { 1, 3, 2, 3, 1 }

0 1 2 3 4

→ reverse pair → arr[i] > 2* arr[j]

Reverse {3, 1} {2, 4}, {3, 4}

Ans-

arr[1] = 3

arr[4] = 1

3 > 2*1

3 > 2

So, first approach is Brute force approach.

Brute force approach: Cond^n $i < j$ & $a[i] > 2 \times a[j]$

- We generate all pair and check the condition.

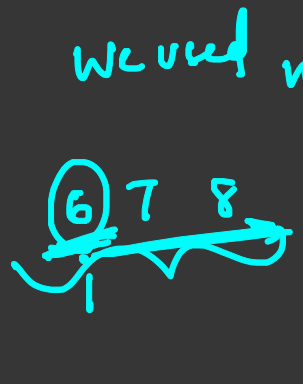
```
i = 0 ——————→ n
    for( j = i+1 ——————→ n)
        if( a[i] > 2* a[j])
            count++.
```

T.C. = $O(N^2)$

↳ nearly $O(N^2)$

Optimized Approach:

- So, idea is to use logic and intuition of count inversion.
  - We used merge sort
  - → $i < j$ && $a[i] > a[j]$



→ Logic was that if $a[i] > a[j]$ then for every $i+1$ the cond^n will be true and count added to the count.

- But we cannot use it in the cond^n as
  - → $a[i] > 2 * a[j]$

Let say for example, 6 1 3 1 2 1 2 3

$6 \not> 2 * 3$ but $13 > 2 * 3$ → we miss this

eg: { 6 13 21 25 } { 1 2 3 4 4 6 9 11 13 }

Idea → is to compare

iterate and find the element which greater (cond^n → both are sorted)

$6 > 1^{st} 2$

$6 > 2^{nd} 2$

$6 \not> 3^{rd} 2$

$6 \rightarrow \{1, 2\}$

$13 \rightarrow \{1, 2, 3, 4, 4, 5\}$

$21 \rightarrow \{1, 2, 3, 4, 4, 5, 9\}$

$25 \rightarrow \{1, 2, 3, 4, 4, 5, 9, 11\}$

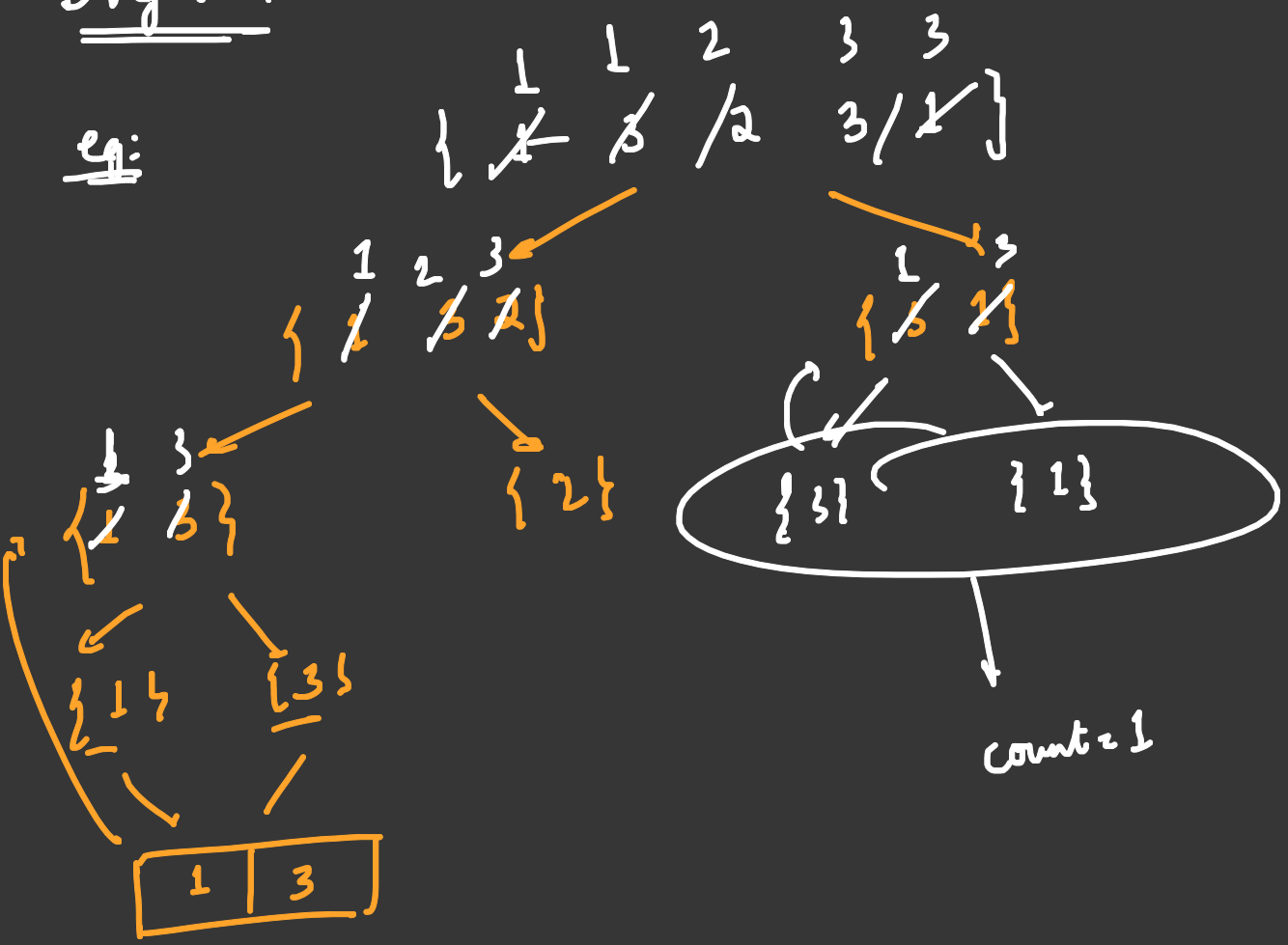
Observe pattern

Because the if the 6 > some value as array is sorted following element bound in array was so greater.

There are repeating no.

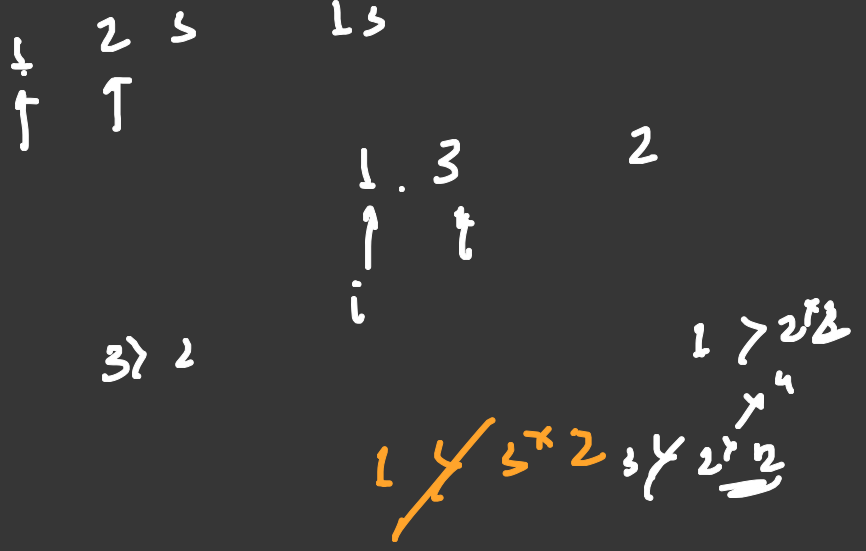
Dry Run:

eg:



We perform merge sort

To divide and conq.



Count = 1+1 = 2

Ans.

Pseudo code

MergeSort func

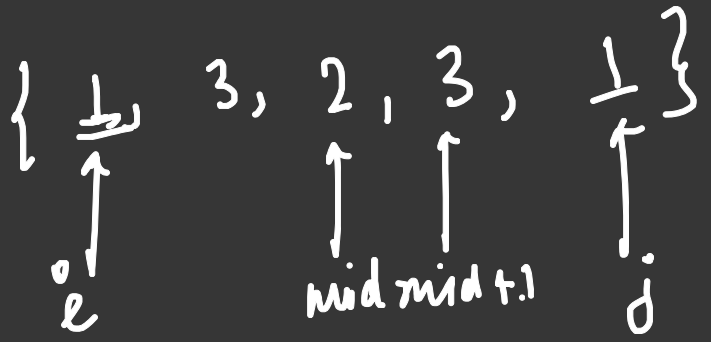
```
f( i, j, arr)
    int count=0;
if( i<j)
    int mid= i+j/2;
    count+= mergeSort( i, mid, arr)
    count+= mergeSort (mid+1, high, arr)
    count+= merge( i, j, mid, arr)
```

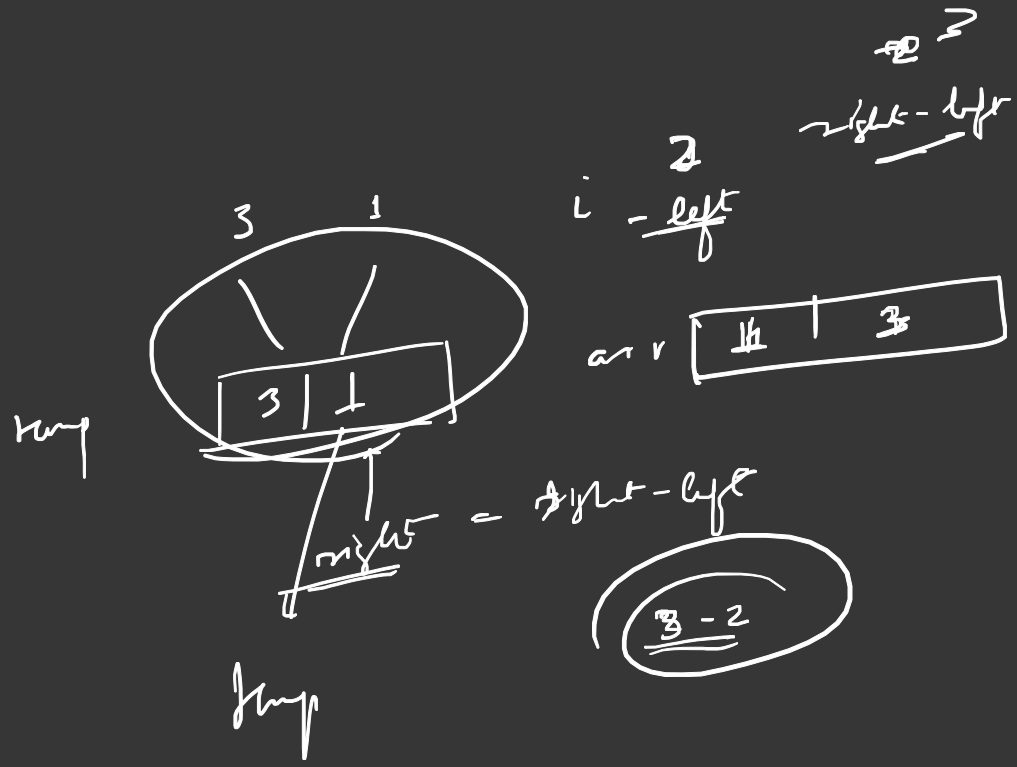
Merge func

```
f( i, j, mid, arr)

for( i= 0 ⟶ mid)

    while( r < n && a[i] > 2*a[j])
        r++;
    count+= r - mid+1
```





$$T.C. = O(\log N (N + N))$$

$$O(2 \log N (2N))$$

$$O(2N \log N)$$

Important:

→ We will have to distort or reorder the array

We can take duplicate array if we have not to change the array.