Reverse pairs

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
```

Example 2:

```
Input: nums = [2,4,3,5,1]

Output: 3
```

Solution 1: Brute Force Approach

Intuition :

As we can see from the given question that **i < j**, So we can just use 2 nested loops and check for the given condition which is arr [i] > 2* arr[j]**.**

Approach:

- We will be having 2 nested For loops the outer loop having i as pointer
- The inner loop with j as pointer and we will make sure that 0 <= i < j < arr.length() and also arr[i] > 2*arr[j] condition must be satisfied.

```
static int reversePairs(int arr[]) {
    int Pairs = 0;
    for (int i = 0; i < arr.length; i++) {
     for (int j = i + 1; j < arr.length; j++) {
       if (arr[i] > 2 * arr[j]) Pairs++;
     }
    }
    return Pairs;  }
```

Time Complexity: O (N^2) (Nested Loops)

Space Complexity: O(1)

Approach 2: Using Merge sort

Solution 2: Optimal Solution

Intuition:

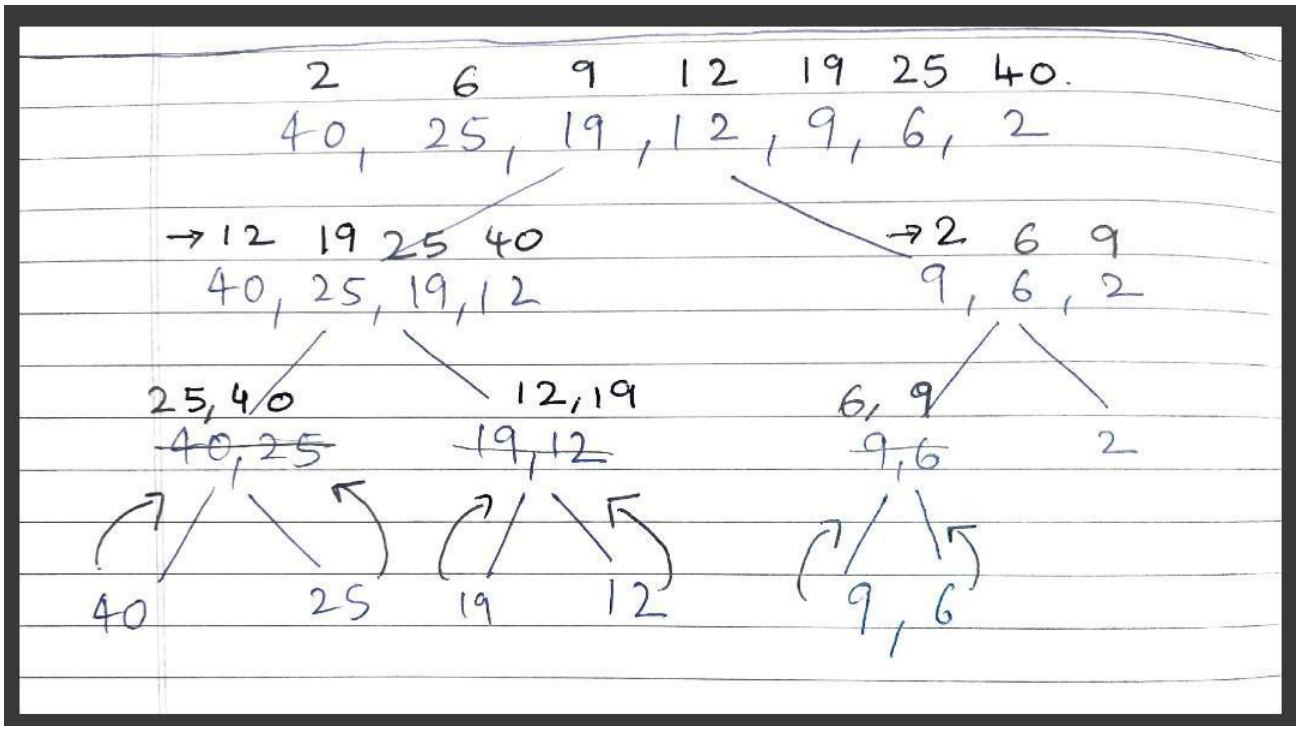
-> We will be using the Merge Sort Algorithm to solve this problem. We split the whole array into 2 parts creating a Merge Sort Tree-like structure. During the conquer step we do the following task :

-> We take the left half of the Array and Right half of the Array, both are sorted in themselves.

-> We will be checking the following condition arr[i] <= 2*arr[j] where i is the pointer in the Left Array and j is the pointer in the Right Array.

-> If at any point arr[i] <= 2*arr[j] , we add 1 to the answer as this pair has a contribution to the answer.

-> If Left Array gets exhausted before Right Array we keep on adding the distance j pointer traveled as both the arrays are Sorted so the next ith element from Left Subarray will equally contribute to the answer.



-> The moment when both Arrays get exhausted we perform a merge operation. This goes on until we get the original array as a Sorted array.

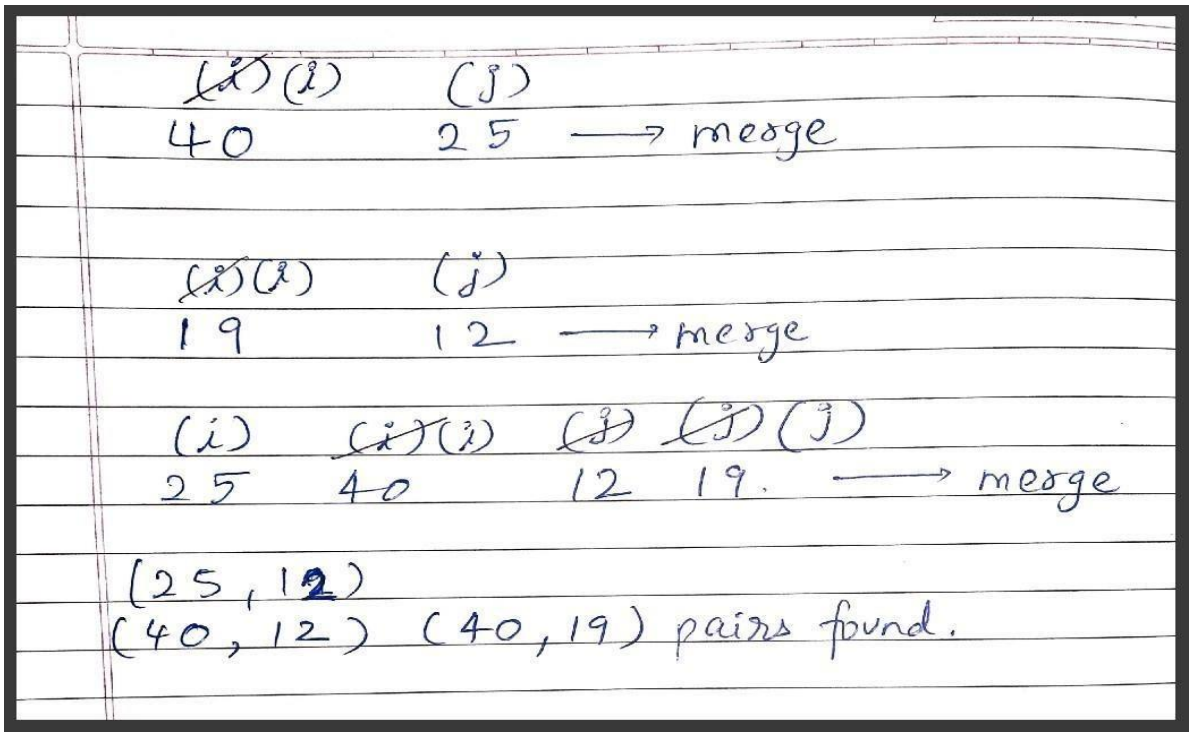
Approach :

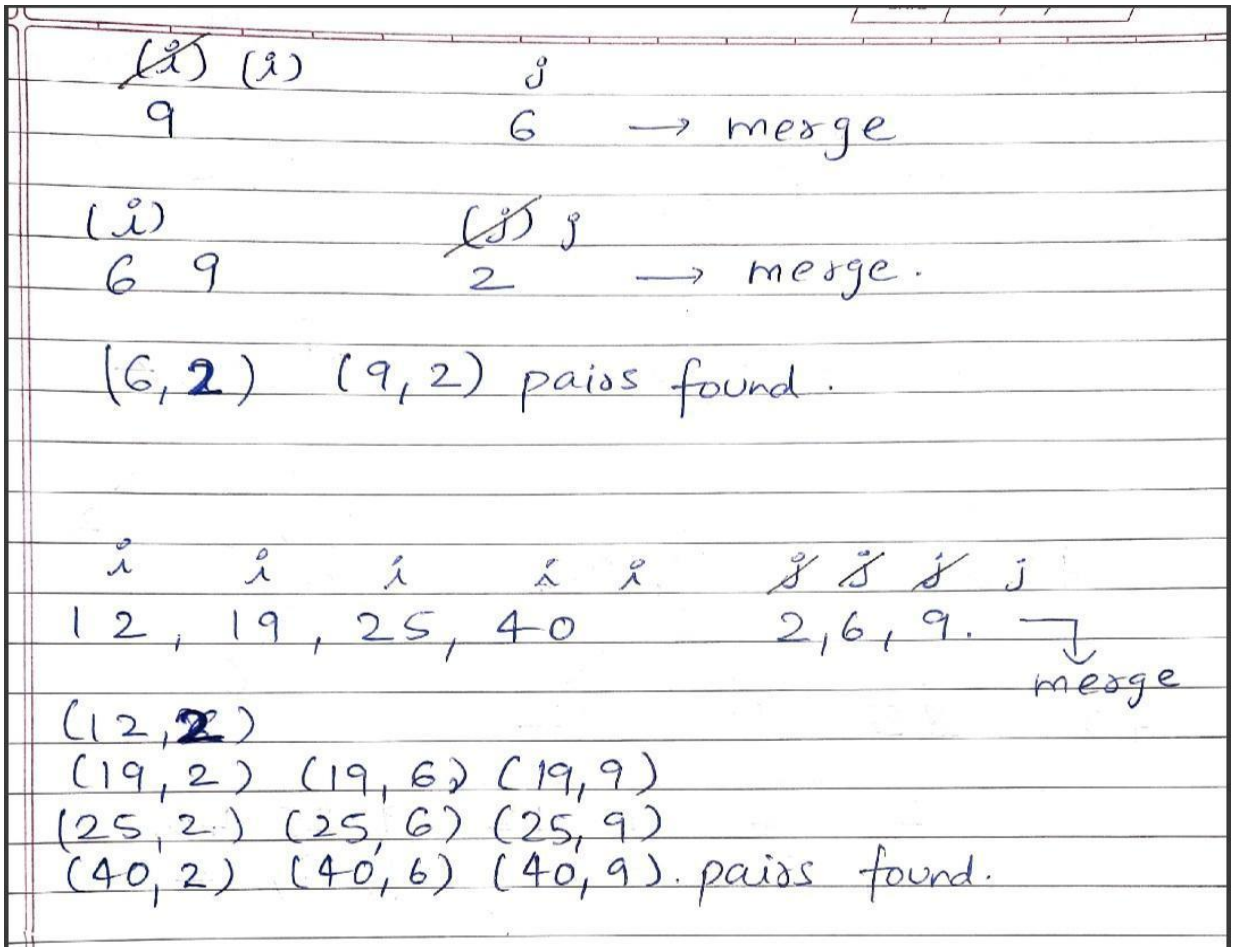
-> We first of all call a Merge Sort function, in that we recursively call Left Recursion and Right Recursion after that we call Merge function in order to merge both Left and Right Calls we initially made and compute the final answer.

-> In the Merge function, we will be using low, mid, high values to count the total number of inversion pairs for the Left half and Right half of the Array.

-> Now, after the above task, we need to Merge the both Left and Right sub-arrays using a temporary vector.

-> After this, we need to copy back the temporary vector to the Original Array. Then finally we return the number of pairs we counted.





Time Complexity : O(N log N) + O (N) + O (N)

Reason : O(N) – Merge operation , O(N) – counting operation (at each iteration of i , j doesn't start from 0 . Both of them move linearly)

Space Complexity : O(N)

Reason : O(N) – Temporary vector

Use the merge sort technique.

```
class Solution {
    public int reversePairs(int[] nums) {
        return merge(nums,0,nums.length-1);
```

```
    }
    public int merge(int[] arr,int l,int r)
    {
        int count=0;
        if(l<r)
        {
            int mid=(l+r)/2;
            count+=merge(arr,l,mid);
            count+=merge(arr,mid+1,r);
            count+=mergesort(arr,l,mid,r);

        }
        return count;
    }
    public int mergesort(int[] arr,int l,int mid,int r)
    {
        int j=mid+1;
        int count=0;
        for(int i=l;i<=mid;i++)
        {
            while(j<=r&&((long)arr[i]>(2*(long)arr[j])))
            {
                j++;
            }
// if a[i] and a[j] form a pair then a[i] and {a[mid+1] to a[j-1]} should also form a pair because
the elements are sorted due to mergesort
```

```
    count+= (j-(mid+1));
}
int[] left=Arrays.copyOfRange(arr,l,mid+1);
int[] right=Arrays.copyOfRange(arr,mid+1,r+1);
int i=0,k=l;
j=0;

while(i<left.length&&j<right.length)
{
    if(left[i]<right[j])
    {
        arr[k++]=left[i++];
    }
    else
    {
        arr[k++]=right[j++];
    }
}
while(i<left.length)
{
    arr[k++]=left[i++];
}
while(j<right.length)
{
    arr[k++]=right[j++];
```

```
        }
        return count;
    }
}
```