↳ Merge Sort variation

Q16 Count inversions in the array
Two elements form an inversion if a[i] > a[j] and i < j. The inversion count indicates how far the array is from being sorted.

i/p → {2, 4, 1, 3, 5}
o/p → 3

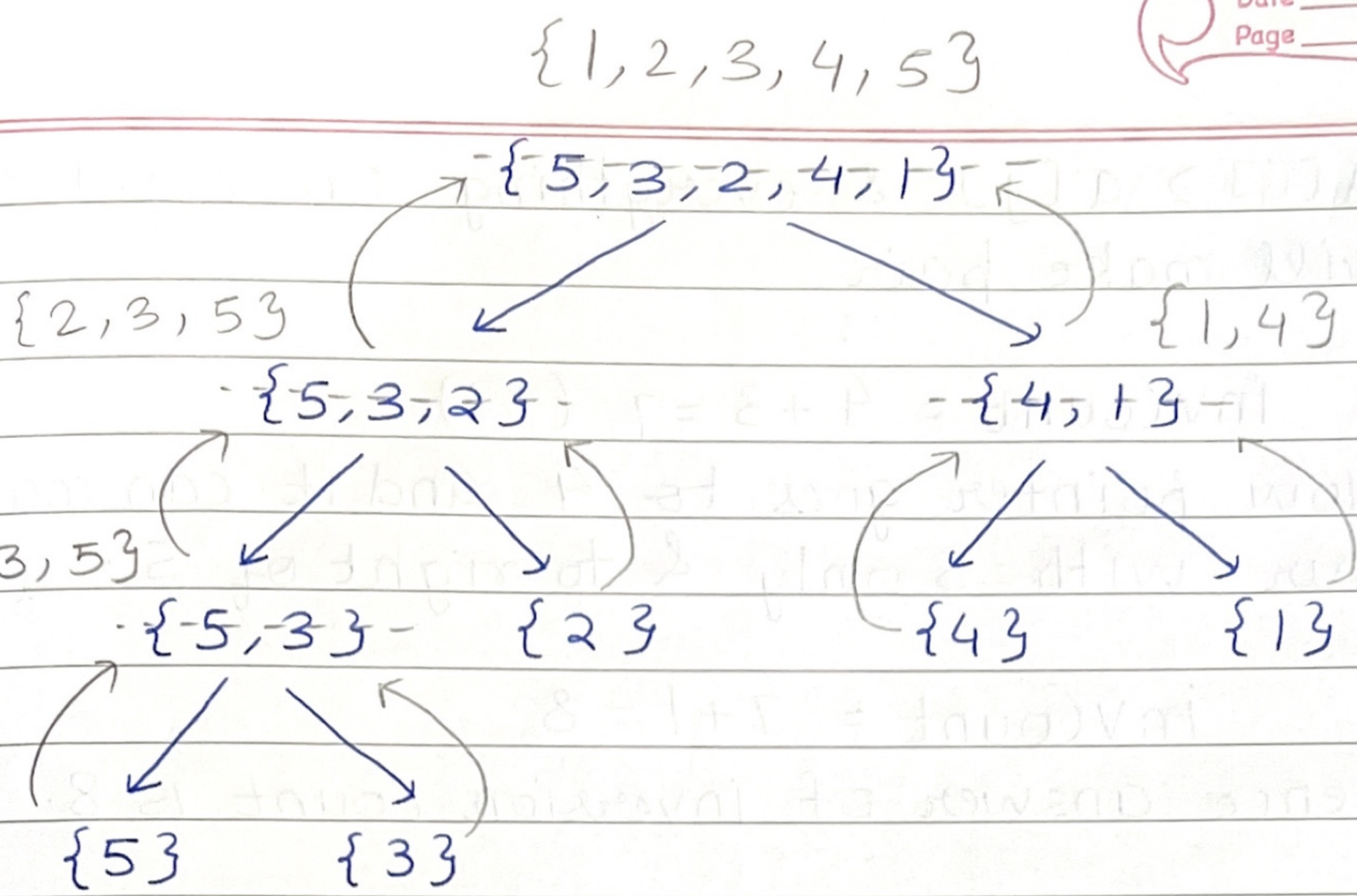
Brute force
Run 2 nested for loops, outer one from 0 to n & inner one from i+1 to n & if we found any element arr[i] > arr[j], then inversion count is increased by 1. Now here we don't check whether i < j or not as j = i+1 } starting & hence is always greater than i.

Time complexity = $O(n^2)$
Space complexity = $O(1)$

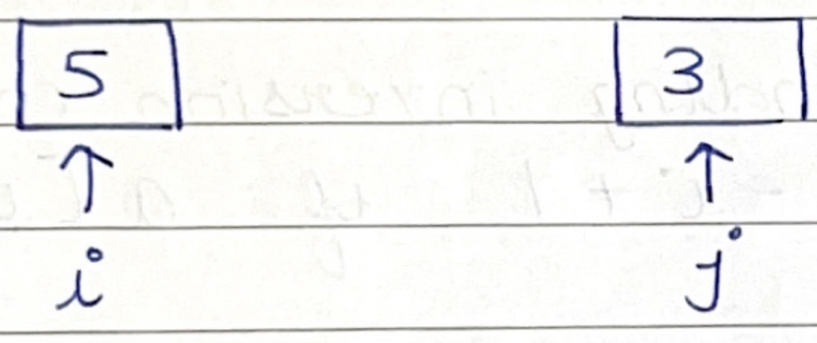
The above solution passed 100/117 test cases on GFG and after that it gave TLE.

Optimal approach
The optimal solution of count inversions can be found out by merge sort.

We can find the inversion count by slightly modifying the merge function of the merge sort.

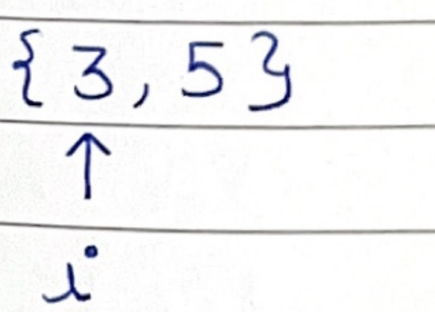


Consider merging of {5} and {3}

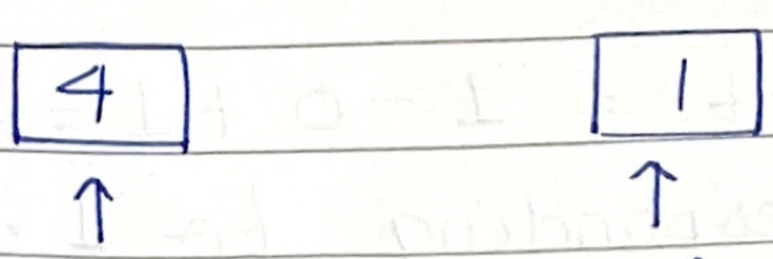


Single element is sorted.

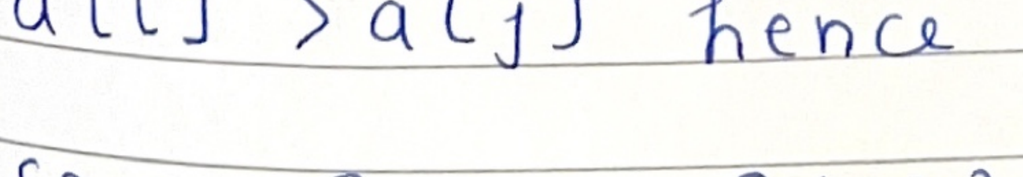
a[i] > a[j] → Hence invCount = 1.



a[i] > a[j] and everything to the right of 3 is greater than 2 due to sorted array.
Hence invCount = 3 (+2)



a[i] > a[j] hence invCount = 4



a[i] > a[j] & everything on right of 2 will make pair

invCount = 4 + 3 = 7 (+3)

Now pointer goes ~~to~~ 4 and it can make pair with 5 only & to right of 5.

invCount = 7 + 1 = 8

Hence answer of inversion count is 8.

Time complexity = O(n logn)

Space complexity = O(n)

Formulae for finding inversion count

invCount = mid - i + 1 if a[i] > a[j].

a[i] > a[j] ⇒ invCount = 1 - 0 + 1 = 2

2 inversion count corresponding to 1.

{4,8} {1,2}

a[i] > a[j] ⇒ invCount = mid - i + 1

= 1 - 0 + 1 = 2

Hence we have derived the formulae for finding inversions for a particular element.

Code

```
long merge (vector <int> & arr , vector <int> &temp,
                long s, long mid, long e) {

    long i = s ; // 1st array starting
    long j = mid+1 ; //2nd array starting
    long k = s ;
    long invCount = 0 ; //No of inversion counts.
    //Run loop until any of the array finishes
    While (i<= mid && j<= e) {
            if (arr [i] <= arr [j]) {
                temp [k++] = arr [i++] ;
        }
        else { //Inversion count case
            temp [k++] = arr [j++] ;
Formulae →  invCount += mid - i + 1 ;
        }
    }
    //If any of array is bigger in size
    While (i<= mid) {
        temp [k++] = arr [i++] ;
    }
    While (j<= e) {
        temp [k++] = arr [j++] ;
    }
    While (s<= e) {
        arr [s] = temp [s] ;
        s++ ;
    }
```

```
    return invCount;
}

long mergeSort (vector<int> & arr, vector
                <int> & temp, long s, long e){
    // Base condition
    if (s >= e) {
        return 0;
    }
    long mid = s+(e-s)/2;
    long invCount = 0;
    // Recursive calls
    invCount += mergeSort (arr, temp, s, mid);
    invCount += mergeSort (arr, temp, mid+1, e);
    invCount += merge (arr, temp, s, mid, e);
    return invCount;
}

long countInversions (vector<int> & arr){
    long invCount = 0;
    vector<int> temp (arr.size(), 0);
    invCount = mergeSort (arr, temp, 0,
    arr.size()-1);
    return invCount;
}
```