left i=0

2	6	12	13	21
0	1	2	3	4

count = 0

j = 0

right

4	5	7	9	10	11
0	1	2	3	4	5

- 2 > 2×4 ⊗ (i++) ② — ① count = count + j = 0 + 0 = 0
- 6 > 2×4 ⊗ (i++) — count = count + j = 0 + 0 = 0
- 12 > 2×4 ✓ (j++)
- 12 > 2×5 ✓ (j++)
- 12 > 2×7 ⊗ (i++) ② — ① count = count + j = 0 + 2 = 2
- 13 > 2×7 ⊗ (i++) ② — ① count = count + j = 2 + 2 = 4
- 21 > 2×7 ✓ (j++)
- 21 > 2×9 ✓ (j++)
- 21 > 2×10 ✓ (j++)
- 21 > 2×11 ⊗ (i++) ② — ① count = count + j = 4 + 5 = 9

```
int j=0, count=0;
for(int i=0; i< left.length ; i++) {
    while( j < right.length && left[i] > 2 * right[j]){
        j++;
    }
    count = count + j ;
}
```

Now, the above question converted in terms of p, q, r like merge (during merge sort)

p				q
2	6	12	13	21

q+1

4	5	7	9	10	r
6	7	8	9	10	→ index

i = p (start)

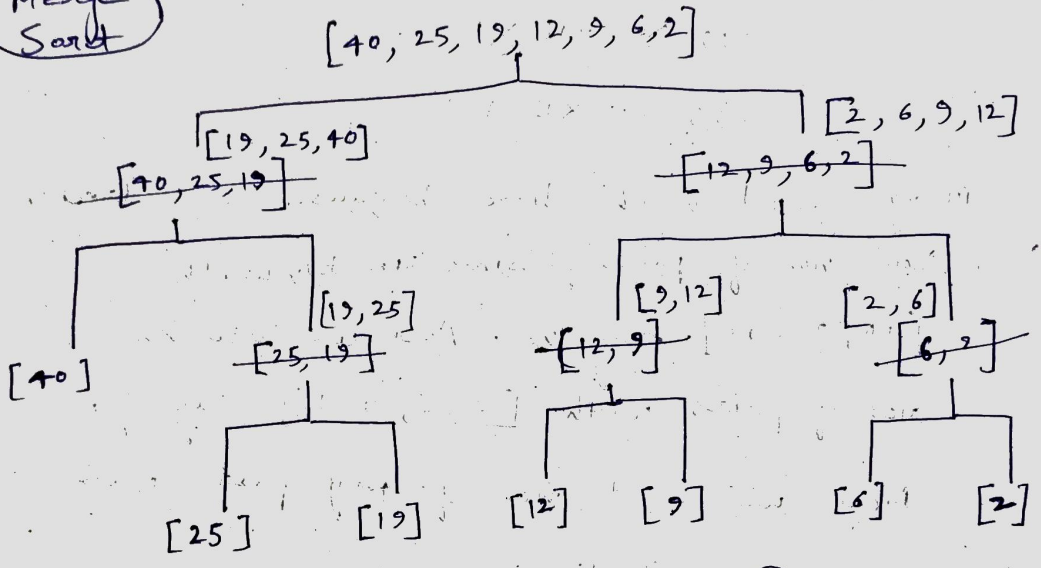
j = q+1 (start)

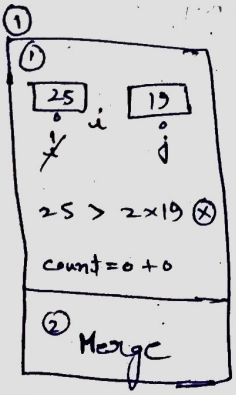
```
int j=q+1, count=0
for(int i=p; i<=q; i++){
    while(j<=r && left[i]>2*right[j]){
        j++;
    }
    count=count+(j-(q+1));
}
```

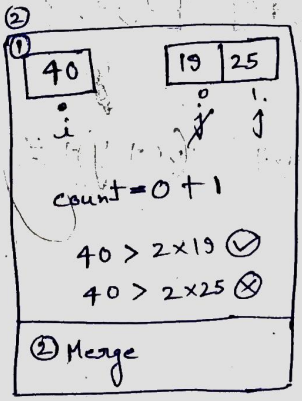
Now, we have an array:

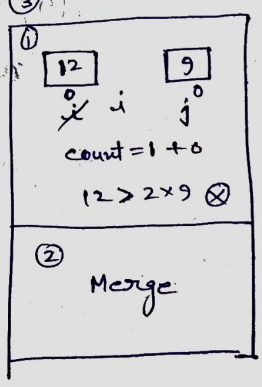
40	25	19	12	9	6	2

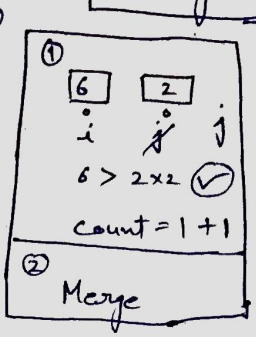
Merge Sort

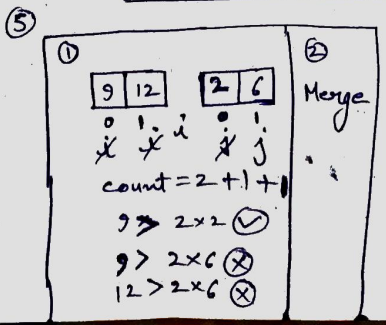


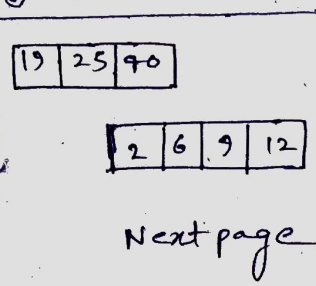




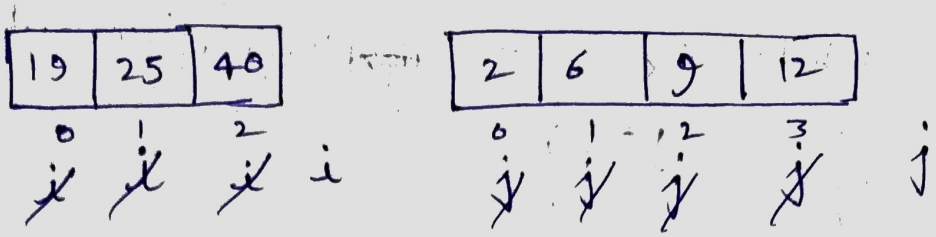








Next page



count = 4 + 3 + 4 + 4 = 15

19 > 2×2 ✓

19 > 2×6 ✓

19 > 2×9 ✓

19 > 2×12 ✗

25 > 2×12 ✓

Merge Sort works here because we already know no. of pairs possible ~~in 19~~ with [19, 25, 40] say x. And we also know no. of pairs with [2, 6, 9, 12] say y.

Now, we want no. of ~~pairs~~ pairs with [19, 25, 40] [2, 6, 9, 12] say z

So, count = (x + y) + z

Length of left sorted array

Length of right sorted array

Time Complexity for our countReversePair() method = O(n1 + n2) = O(n)
So, total time complexity = O(log n) * (O(n) + O(n1 + n2)) = O(log n) * (O(n) + O(n))
T = O(log(n)) * O(2n)

T = O(n*log(n))

Merge

CountReversePair Method