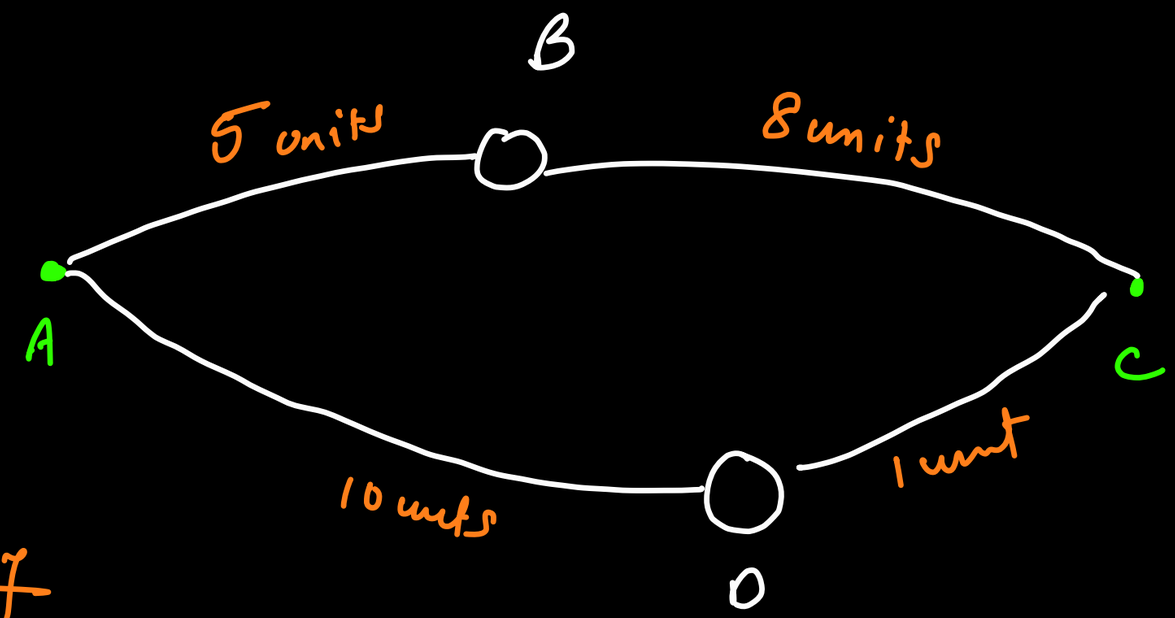
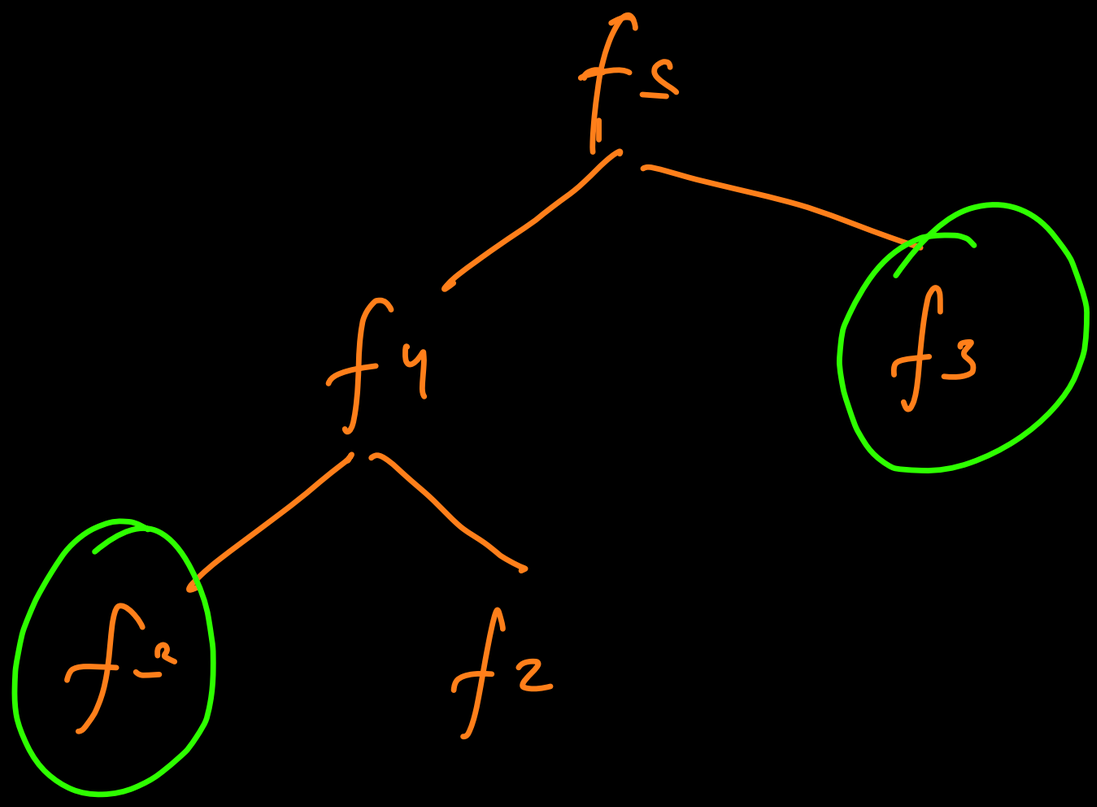
Algorithm Paradigms

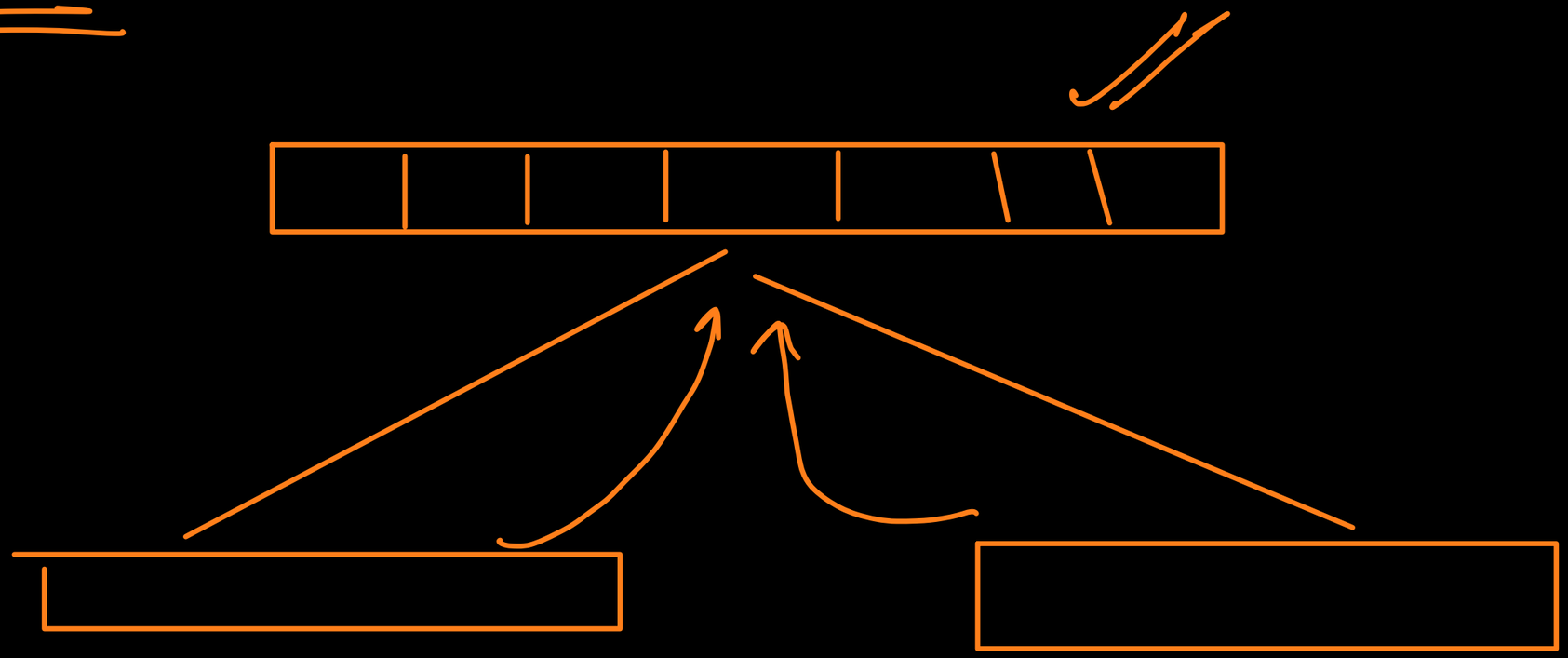
1. Brute force
2. Greedy
3. Dynamic Programming
4. Divide N Conquer.

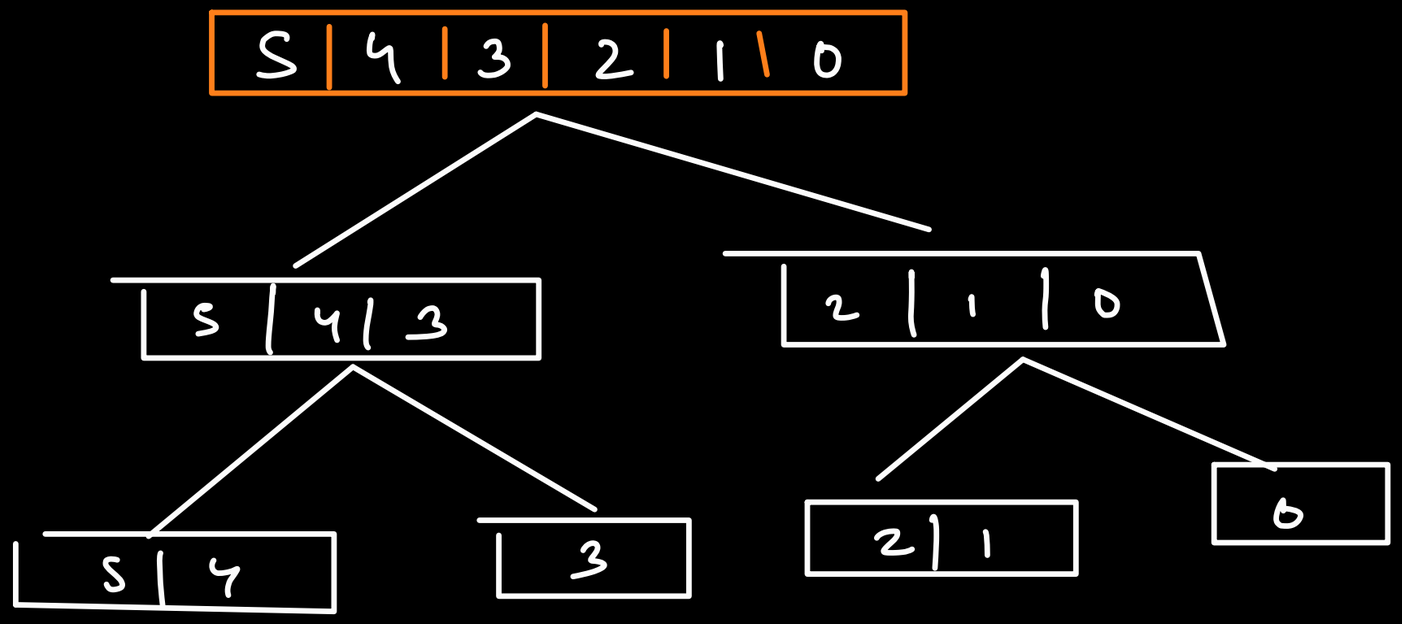




Divide N Conqu (DNC)

Merge Sort





$$T(n) = a \times T\left(\frac{n}{b}\right) + f(n)$$

$T(n) = T(n/2) + T(n/2) + O(n)$

$T(n)$ ↓ no. of operations reqd to do merge sort on an array of length n.

$T(n/2)$ ↓ no. of ops to sort the left half

$T(n/2)$ ↓ no. of ops to sort the right half

$$\Rightarrow T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$O(n+m)$

level	Size of problem	# of problems	Ops
0	n	1	n
1	$\frac{n}{2}$ $\frac{n}{2}$	2	n
2	$\frac{n}{4}$	4	n
3	$\frac{n}{8}$	8	n
⋮	⋮	⋮	⋮
i	$n/2^i$	2^i	n
⋮	⋮	⋮	⋮
K ↳ last level	1	2^K	n

$K = \log_2 n$

$2^{\log_2 n} \rightarrow n$

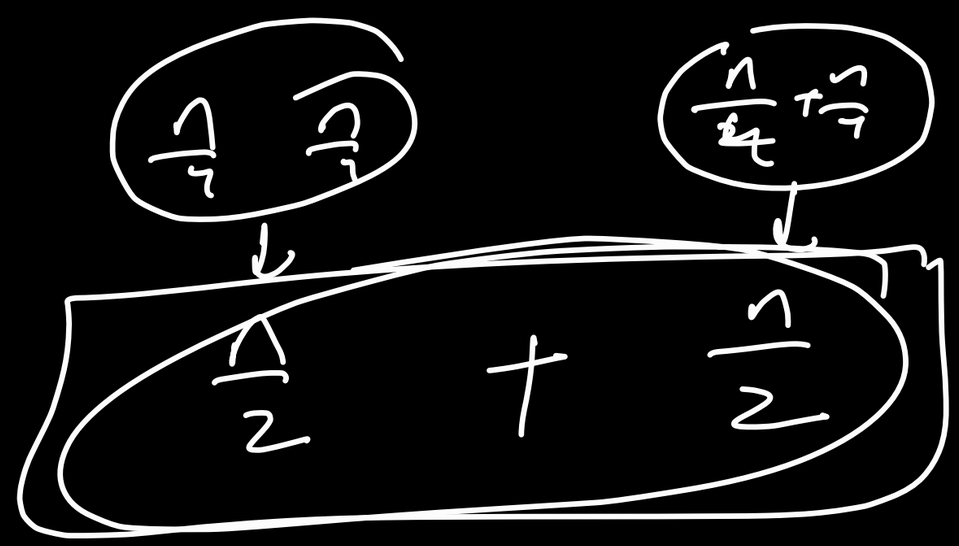
$$\frac{n}{2^k} = 1$$

$n = 2^k$ taking log both sides

$$\log_2 n = \log_2 2^k$$

$$\log_2 n = k \log_2 2$$

$$\boxed{k = \log_2 n}$$



$T(n) = n + n + n \cdots\cdots n$ ($\log_2 n$ times)

$T(n) = n \log n$

$O(n \log n)$

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

$f(n) = O(n^d)$ → work done on each level

$a \rightarrow$ no. of subproblems

$\frac{n}{b} \rightarrow$ size of each subproblem

$a > 0$ $\quad b > 1$ $\quad d \geq 0$

Master theorems

$$T(n) = a\,T\left(\frac{n}{b}\right) + O(n^d)$$

levels	Size	# of probs	ops
0	n	1	$O(n^d)$ —
	↓		
1	$\frac{n}{b}$	a	$a \times O\left(\frac{n}{b}\right)^d$ —
2	$\frac{n}{b^2}$	a^2	$a^2 O\left(\frac{n}{b^2}\right)^d$ —
⋮	⋮	⋮	⋮
i	$\frac{n}{b^i}$	a^i	$a^i \times O\left(\frac{n}{b^i}\right)^d = O(n^d) \times \left(\frac{a}{b^d}\right)^i$
⋮	⋮	⋮	⋮
$\log_b n$	1	$a^{\log_b n}$	$\rightarrow a^{\log_b n} \times 1$

Total ops $\Rightarrow$ $\sum_{i=0}^{\log_b n} O(n^d)\left(\frac{a}{b^d}\right)^i$

$\downarrow$

$O(n^0) \times \left(\frac{a}{b^d}\right)^0 + O(n^d)\left(\frac{a}{b^d}\right)^1 + O(n^d)\left(\frac{a}{b^d}\right)^2 \cdots$

$a \to O(n^0)$

$r \to \left(\frac{a}{b^d}\right)$

$\hookrightarrow$ ~~GP~~

$a + ar + ar^2 \cdots\cdots ar^{n-1}$

$r < 1$

$\hookrightarrow$

$r < 1 \rightarrow a \times \frac{(-r^n)}{1-r} \rightarrow O(a)$

$r = 1 \rightarrow a+a+a \cdots a$ (n times) $= na$

$r > 1 \rightarrow a \times \frac{(r^n - 1)}{r-1} \rightarrow O(ar^n)$

Case 1 $\frac{a}{b^d} < 1 \rightarrow a < b^d \rightarrow d > \log_b a$

$r < 1$

↳ Time → $O(n^d)$

Case 2 $\frac{a}{b^d} = 1 \rightarrow a = b^d \rightarrow d = \log_b a$

$$\sum_{i=0}^{\log_b n} O(n^d) \left(\frac{a}{b^d}\right)^i$$

$$\hookrightarrow \sum_{i=0}^{\log_b n} O(n^d)$$

$$= (1 + \log_b n) \times O(n^d)$$

$$= \underline{O(n^d \log n)} =$$

$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

$d = 1$

$a = 2$

$b = 2$

$\frac{a}{b^d} \rightarrow \frac{2}{2^1} = 1$ =

$O(n \log n)$

Case 3 $\quad \frac{a}{b^d} > 1 \rightarrow a > b^d \rightarrow d < \log_b a$

↳ $\sum_{i=0}^{\log_b n} O(n^d)\left(\frac{a}{b^d}\right)^i$

↳ $O\left(O(n^d)\left(\frac{a}{b^d}\right)^{\log_b n}\right)$

$\rightarrow O\left(n^d \times \frac{a^{\log_b n}}{b^{d \log_b n}}\right) \rightarrow O\left(n^d \times \frac{n^{\log_b a}}{n^d}\right)$

$\rightarrow O(n^{\log_b a})$ ⟸

Let A[0...n - 1] be an array of n distinct positive integers. If i < j and A[i] > A[j] then the pair (i, j) is called an inversion of A. Given n and an array A your task is to find the number of inversions of A.

length of array

$n \leq 10^6$

$A \rightarrow [\ 2, 3, 8, 6, 1\ ]$

ans → 5

(2,1) (3,1) (8,1) (6,1)
(8,6)

→ count inversion pairs

Inversion pair

$(a[i], a[j])$

$i < j$ and $a[i] > a[j]$

↳ $O(n \log n)$

$10^6 \times 20$

$\Rightarrow 2 \times 10^7$ ✓

Brute force → let's try to find all pairs of elements in the array & filter the inversion pair

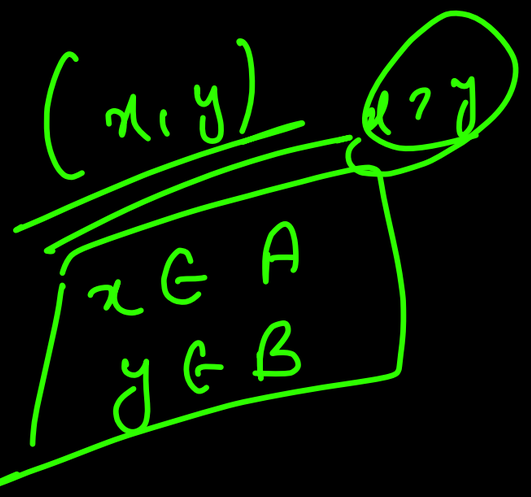
$n \rightarrow O(n^2)$ → TLE

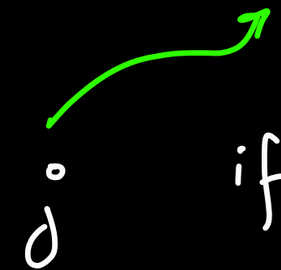
Merge Sort

A

B

} merge the 2 sorted arrays



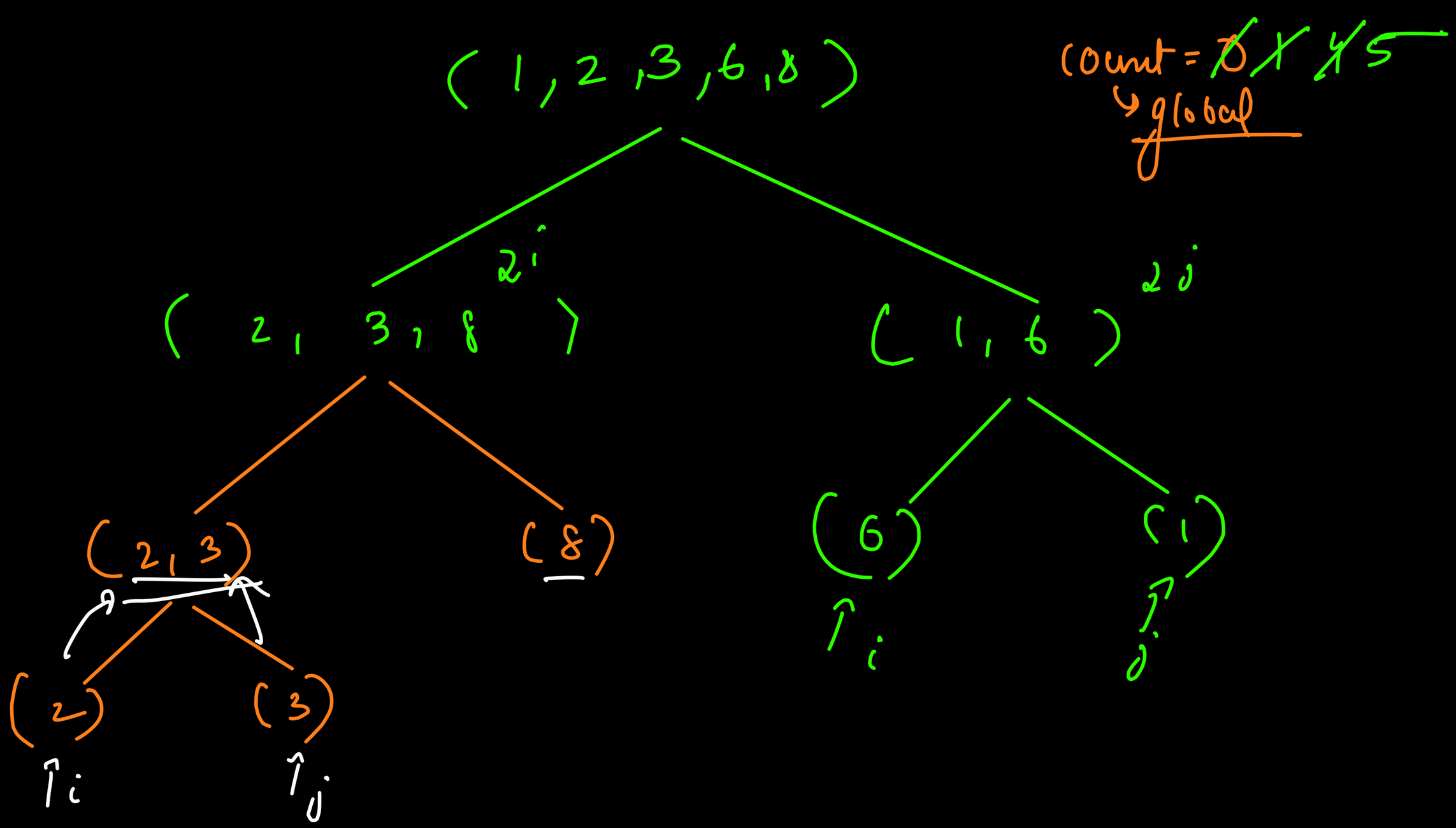



```
if ( A[i] < B[j] ) {
    c[k] = A[i];
    i++; k++;
} else {
    count += (A.length - i)
    c[k] = B[j];
    j++; k++;
}
```

C → | 2 | 5 | 6 | 8 | 10 | 11 | 13 | 14 | 15 | 18 | 19 |

count += (A.length − i)

count = 0 + 6 + 5 + 2 + 2 → 15



$f(arr, i, j)$ =

$f(arr, i, mid)$

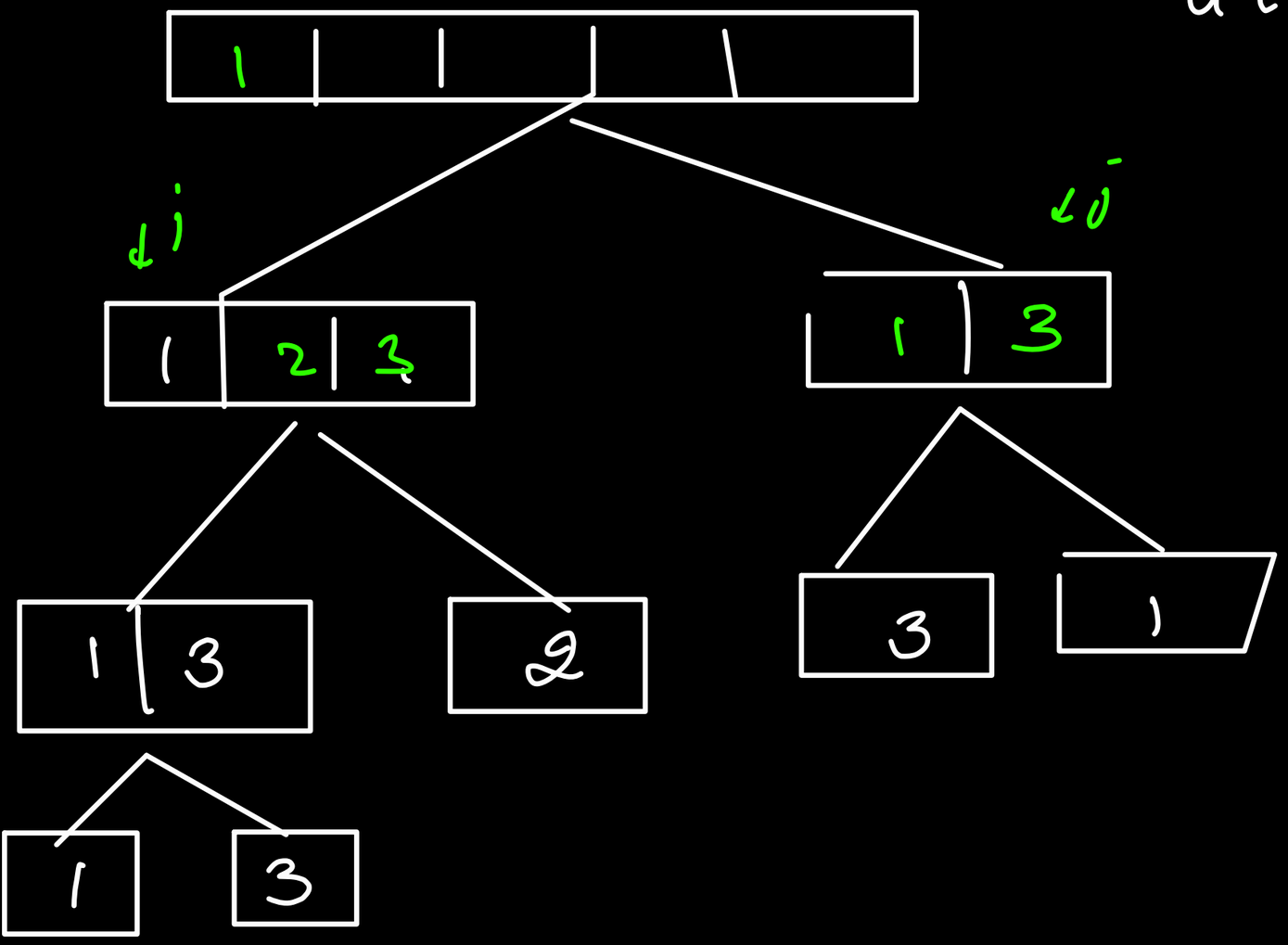
$f(arr, mid+1, j)$

$merge()$

↓

This func^n does mergesort in the range i, j, and adds the inversion pair count of range i, j in count variable.

count = 0 | 1



a[i] > 2 × a[j]

i < j

(x, y)

x ∈ left

y ∈ right

if (a[i] > 2 * b[j])

count ⇐

C →

Comb → 4 + 2

n

A

C

D

S → 2 + 2

reverse pair (x, y)

x ∈ A
y ∈ B

C = | 1 | 1 | 2 | 3 |

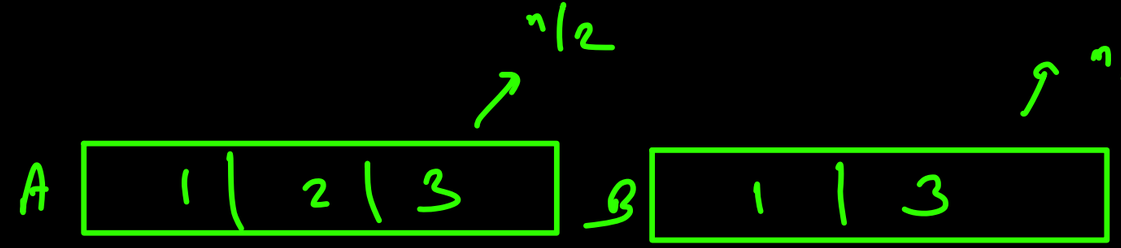
A | 1 | 2 | 3 | B | 1 | 3 |

i

j

```
if (a[i] < b[j])
```

```
else {
    if (a[i] > 2 * b[j]) {
        cnt++;
    }
}
```



$O(n^2)$

TLE

Brute force → all possible pairs (x, y)

$x \in A$

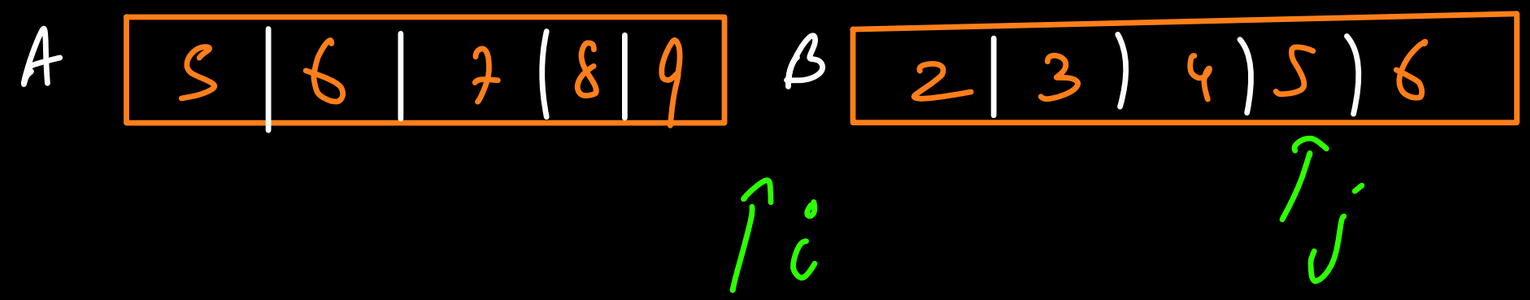
$y \in B$

$x > 2 * y$

count (x, y)

x ← A y ← B

x > 2*y




```
if (A[i] > 2 * B[j]) {
    count += (A.length - i) ;
    j++;
} else {
    i++;
}
```

$i < j$

nums1 , nums2 →

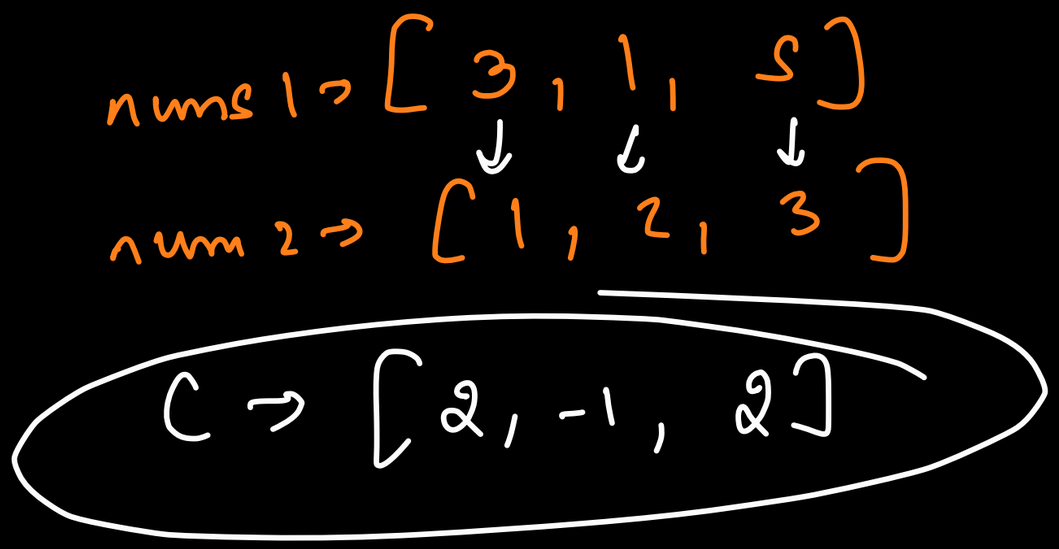
$nums1[i] - nums1[j] <= nums2[i] - nums2[j] + diff$

$\Downarrow\Uparrow$

$num1[i] - num2[i] <= nums1[j] - nums2[j] + diff$

$c[i] <= c[j] + diff$

$c[k] \rightarrow nums1[k] - nums2[k]$



$$i < j$$

$$c[i] <= c[j] + diff$$

$2 < 3$

$$-c[i] >= -c[j] - diff$$

$-2 > -3$

Say → $-c[k] = f[k]$

$$f(i) >= f(j) - diff \quad i<j$$

[3, 2, 5] [2, 2, 1]

C → [1, 0, 4]

diff = 1

f → [-1, 0, -4]

(i < j) fi >= fj - diff

diff = 1

[-1, 0, -4]

count = 1 + 2

2 i

a [-1, 0]

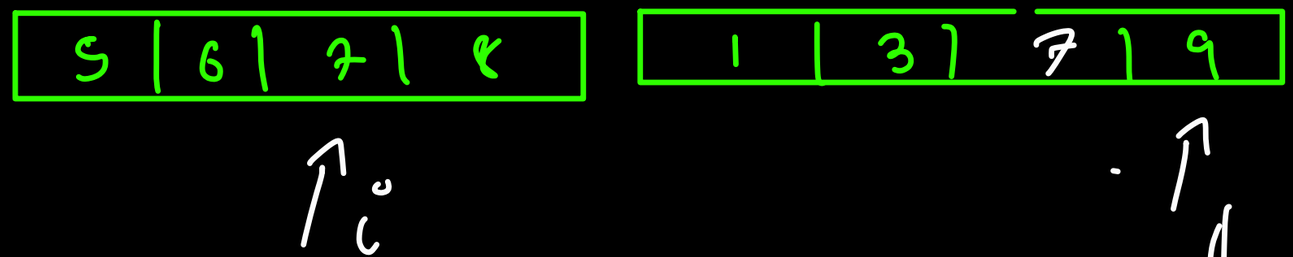
b [-4]

j

(-1)

(0)

diff = 1




```
if ( a[i] >= a[j] - diff) {
    count += (a.length - i)
                    diff
} else {
    j
```

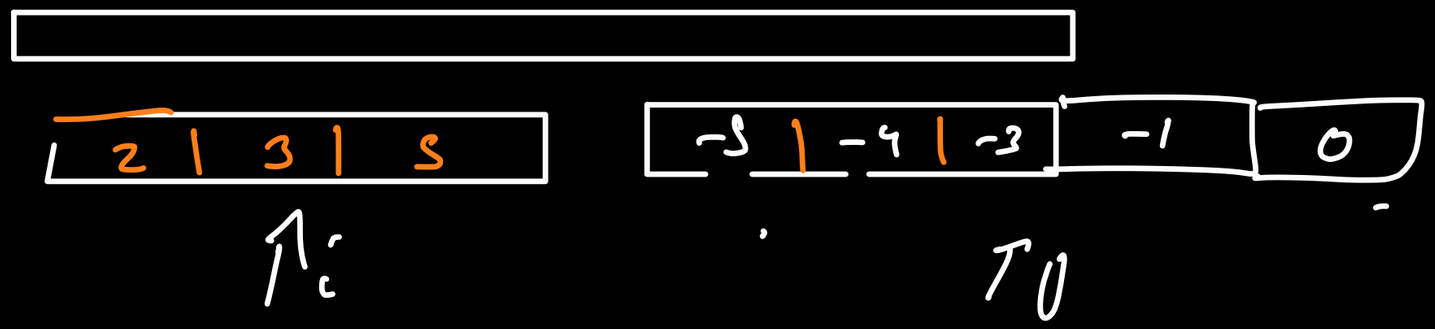
Pair of topics

↳ $a_i + a_j > b_i + b_j$ i ≠ j

$a_i - b_i > b_j - a_j$

$a_k - b_k = c_k$

$c_i > -c_j$



if $(C_i > -C_j)$ {
count += (a.length - i)
} else {
j++
}

$(2, 3, 5) \in -4, 1, 3$

$C_i > C_j$