* Merge Two Sorted Arrays :→ (Two Pointer Approach)

i < n1 a1 = {1, 3, 5, 7, 9} → remaining part ↓ copy

j < n2 a2 = {2, 4, 6} → out of loop

i = 0, j = 0, index = 0;

if (a1[i] < a2[j])

a3[index++] = a1[i++]

else (a2[j] < a1[i])

a3 = {1 2 3 4 5 6 7 9} (0 1 2 3 4 5 6 7)

If any remaining part is present in any of the arrays, just copy/paste.

Recursion Merge Sort Algorithm :→ Divide & Conquer

logN Split (s, m) (m+1, e)

n1 = right length (m − s + 1) = 2 − 0 + 1 = 3

mid = s+e/2

left length (e − m) = n2 = 4/2 = 2

4 − 2 = 2

Sorted → Single Elements

Left side Sorted Array ← Right side Sorted Array

Merging n elements → n (log N)

Merge Sort Time Complexity :→

Spliting the array → Repititive Division by finding mid → O(log n)

Merging the two sorted arrays → O(n)

Total Time Complexity → O(n) × O(log n) → Big O (n log n)

Space Complexity :→ We use two separate arrays to store the left & right part of total 'n' elements → O(n)

Space Complexity :→

* If you are ever taking any extra array or any collections like stack or queue to perform some operations on the data set given, it will be applied on all the 'n' elements of the data set. Therefore Space Complexity becomes → Big O(n).

* Only variables → O(1)

Merge Sort Most Important Interview Question :→

n log n ← Inversion Count | TCS | Accenture | Capgemini | Infosys

int arr[] = {5, 3, 2, 4, 1}

i < j and arr[i] > arr[j]

5 3 2 4 1 (i < j) a[i] > a[j]

(8) 5,3 5,2 5,4 5,1 3,2 3,1 2,1 4,1

→ if a1[i] < a2[j] fae < sae

a3[k++] = a1[i++] fae > sae

2, 3, 5 1, 4, 6 (4) ic

0 1 2 3 4 5

left = [2, 3, 6] → sorted

right = [1, 5, 7] → sorted

ic = 0 arr[i] > arr[j]

2 > 1 → ic

inc += (mid − i + 1;)

= 2 − i + 1

= (3)

→ Wave Sort :→ 10, 90, 49, 2, 1, 5, 23

O/p :→ 2 1 10 5 49 23 90

2 10 49 90 OOB

1 5 23

i = i+2 i = i+2 i = i+2

Sort → 1 2 5 10 23 49 90

Swap Adjacent & jump (i += 2) → 2 1 10 5 49 23 90

Linked Lists :→