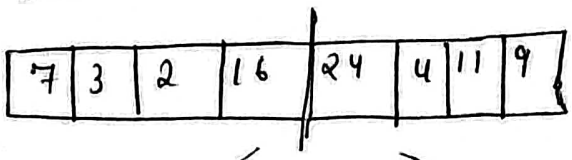
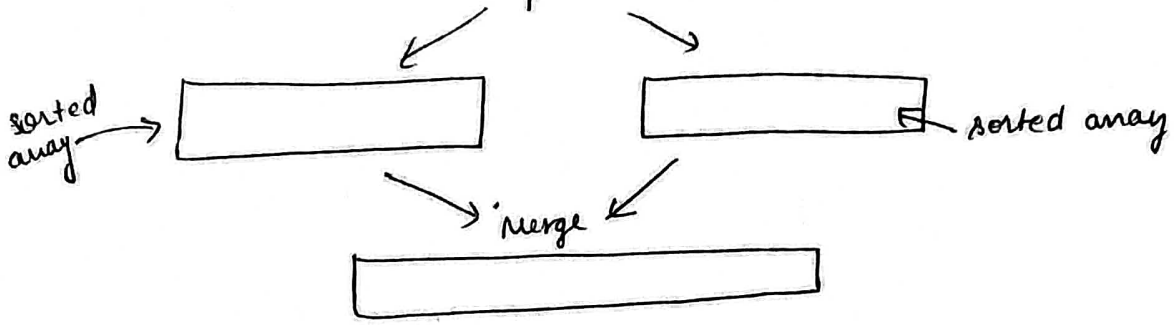
Divide & Conquer

① **Merge Sort** → based on divide & conquer.

i/p →



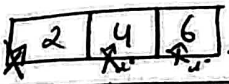


Steps →

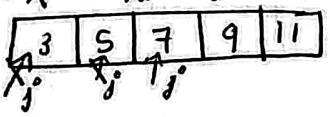
1. Find mid
2. Break into 2 halfs
3. Recursion → 2 halfs → sort them
4. Merge 2 halfs.

Q → Merge two sorted arrays.

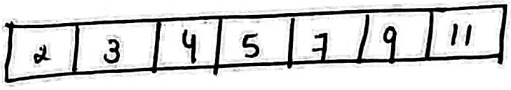
↳ i/p → Array 1 →



Array 2 →



o/p →



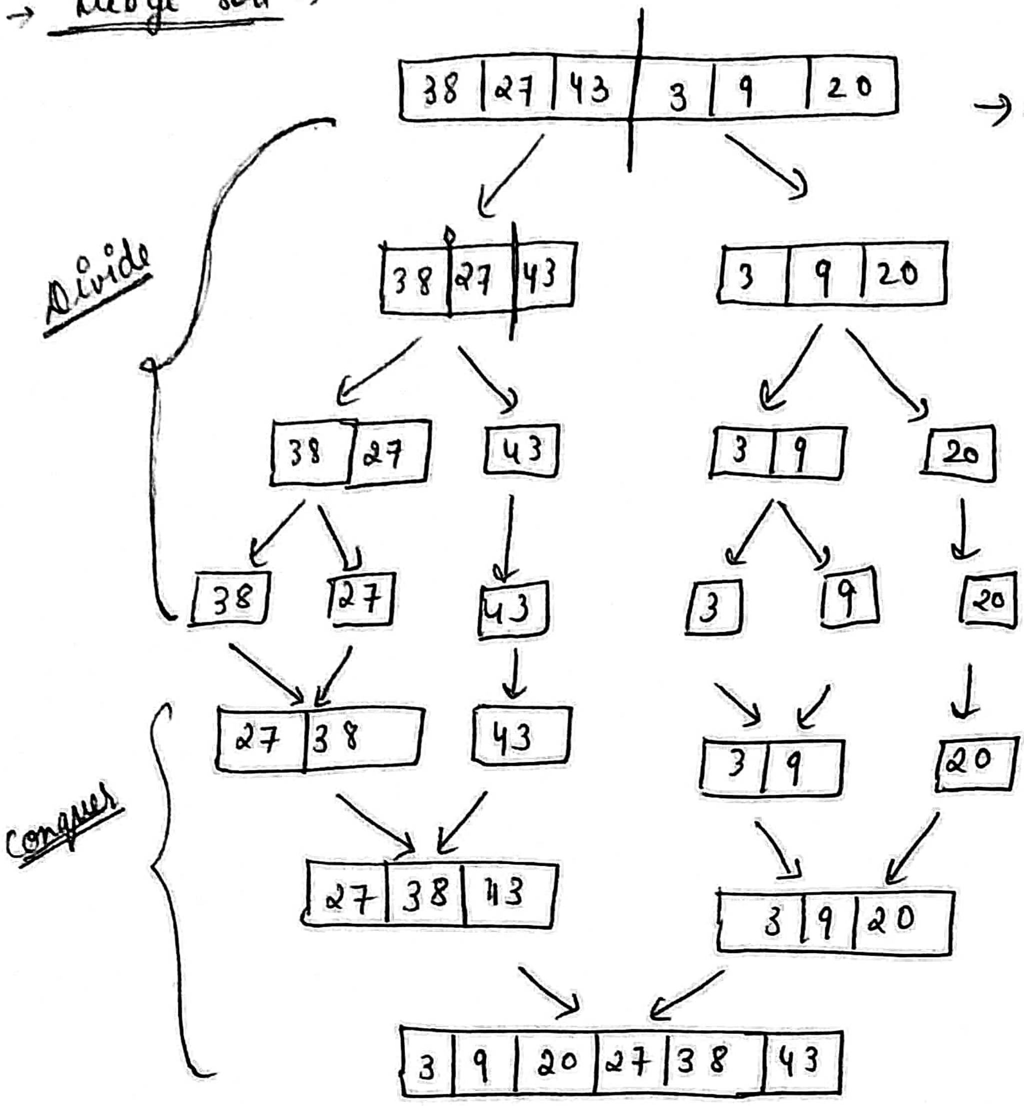
Steps →

1. Two pointers → one at the start of 1st array and one at the start of 2nd array.
2. Compare the elements, whichever element is small push it into ans vector and increase the pointer by 1.
3. If any one array is exhausted then copy other array's remaining elements into the ans vector.

⇒ if (array1[i] < array2[j])
store array1[i] and i++

if (array1[i] > array2[j])
store array2[j] and j++

→ Merge Sort →

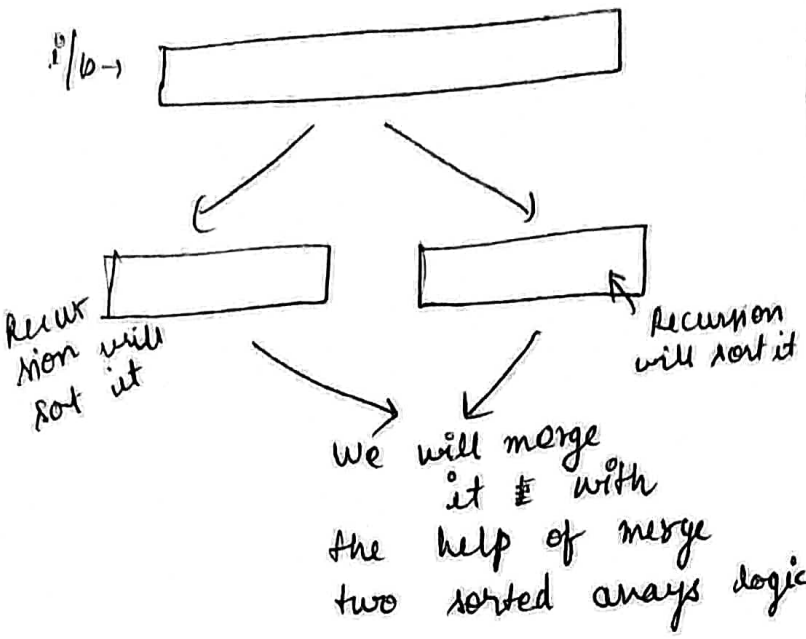


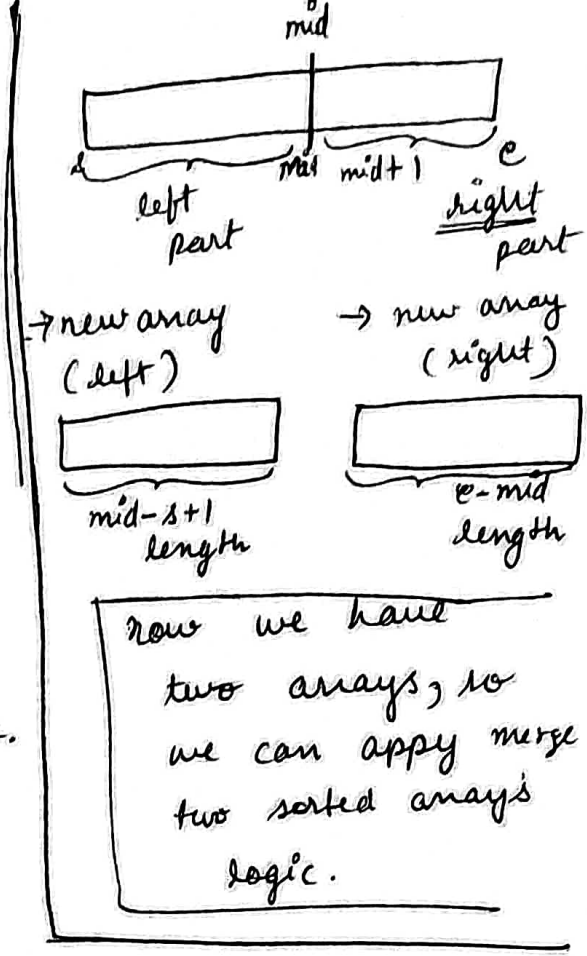
→ break the array into two parts until each part has only one element left. (because one element in any array is already sorted).

→ Merge the array.

```
int main(){
    int arr[] = {4, 5, 13, 2, 12};
    int n = 5;
    int s = 0, e = n-1;
    MergeSort (arr, s, e);

}

void mergeSort (int* arr, int s, int e){
    //base cases ──→ ① Single element    s==e
                └─→ ② Invalid           s>e

    if(s >= e)
        return;
    int mid = s+ (e-s)/2;
    mergeSort (arr, s, mid);       ← call for left half and sort.
    mergeSort (arr, mid+1, e);     ← call for right half and sort.
    merge (arr, s, e);             ← Merge both the arrays.
}
```

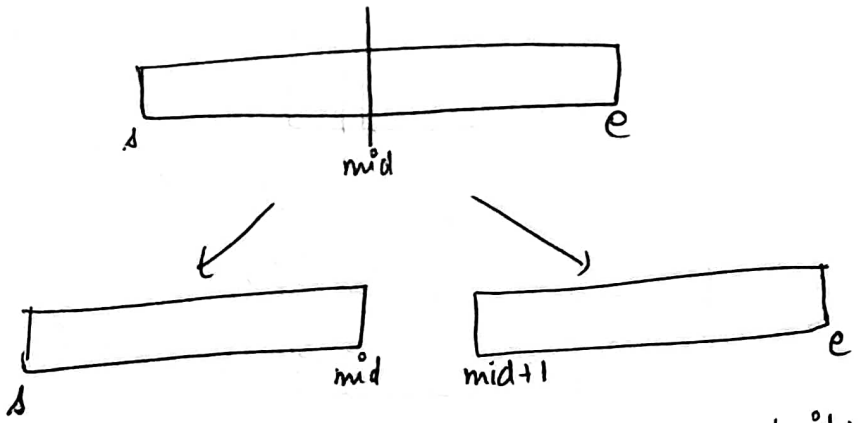
~~void merge(int *arr, int s, int e){~~






```
void merge(int * arr, int s, int e){
    int mid = s + (e-s)/2;
    int len1 = mid - s + 1;
    int len2 = e - mid;
    int* left = new int[len1];
    int* right = new int[len2];   } → Creating arrays dynamically of given lengths.
    // copy values →
    int k = s; ← starting index of left array.
    for(int i = 0; i< len1; i++){
        left[i] = arr[k];
        k++;
    }
    int k = mid + 1; ← starting index of right in array.
    for(int i = 0; i< len2; i++){
        right[i] = arr[k];
        k++;
    }
    // Merge logic →
    int leftIndex = 0;
    int rightIndex = 0;
    int mainArrayIndex = 0;
```

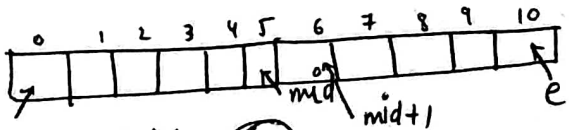
```
while ( leftIndex < len1 && rightIndex < len2){
    if ( left[leftIndex] < right[rightIndex]){
        arr[mainArrayIndex] = left[leftIndex];
        mainArrayIndex++;
        leftIndex++;
    }
    else {
        arr[mainArrayIndex++] = right[rightIndex++];
    }
}
// if right array is exhausted, then we are left with only left array's elements,
while (leftIndex < len1){                    so we will copy them.
    arr[mainArrayIndex++] = left[leftIndex++];
}
// copy right arrays remaining elements.
while (rightIndex < len2){
    arr[mainArrayIndex++] = right[rightIndex++];
}
}
```



length = mid − s + 1

length = e − (mid + 1) + 1
= e − mid − 1 + 1
= e − mid

for eg,



→ length = 5 − 0 + 1 = 6

→ length of right array

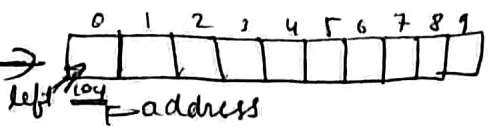
⇒ e − (mid + 1) + 1

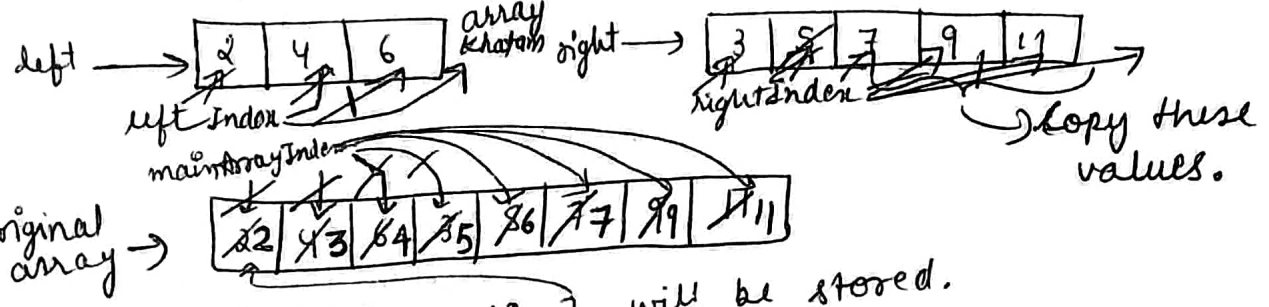
⇒ e − mid = 10 − 5 = 5

int * left = (new) int [10];

keyword (returns address)

That address (returned by new) is stored in left.





2 < 3 → T → so 2 will be stored.

4 < 3 → F → so 3

4 < 5 → T → so 4

5 < 5 → F → so 5

6 < 7 → T → so 6

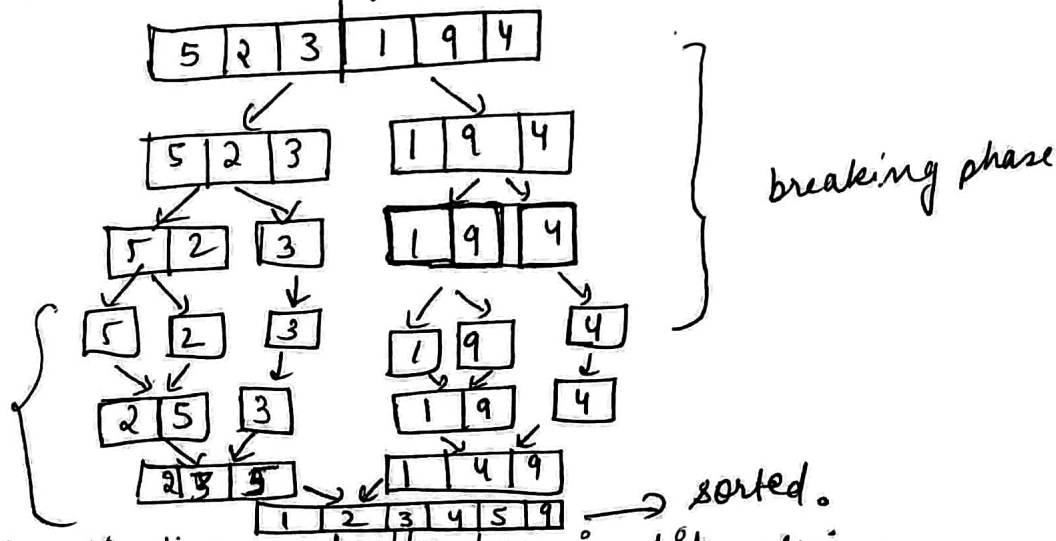
now left array is exhausted, so all the elements of right array will be copied to the array.

Steps →

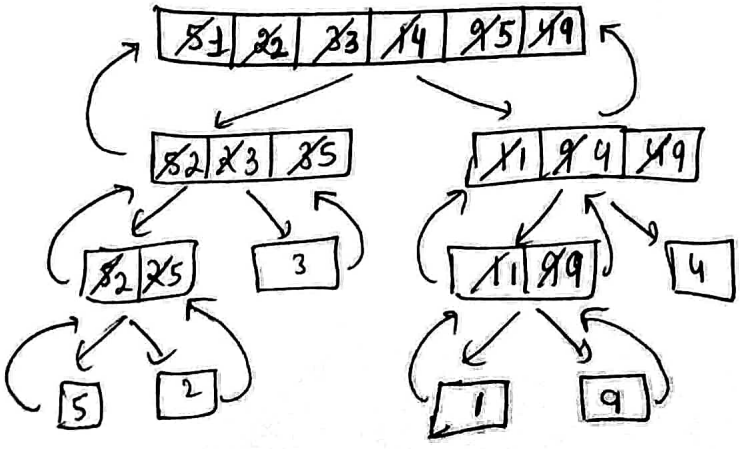
① Compare each element of both array represented by leftIndex & rightIndex until one of array exhausts.

② Copy rem. value from left array into main array.

③ Copy rem. " " right " " " "



→ just for visualization, actually tree is like this →



T.C → O(n log n) ⟶ Find out how?

S.C → O(n) ⟶ Find out how?

Can we ~~use merge~~ implement merge sort without using extra space?

↳ Yes, Inplace Merge Sort → Explore it.

~~V.Imp~~ **H.W - Inversion Count.**

~~① Quick Sort →~~

② **Merge Sort T.C →**

```
mergeSort ( ){
    // BC

    // left call

    // right call

    // merge sort
}
```

left call ↓, right call ↓, merge array ↓

$T(n) = k_1 + T(n/2) + T(n/2) + n*k$

$= k_1 + 2T(n/2) + n*k$

$$\boxed{T(n) = 2T(n/2) + n*k}$$

Solving this eq^n by ~~elimination~~ substitution method →

a times / (a-1) times:

$T(n) = 2T(n/2) + n*k$

$2T\left(\frac{n}{2}\right) = 2\times2\,T\left(\frac{n}{4}\right) + 2\times\frac{n}{2}*k$ × 2

$4T\left(\frac{n}{4}\right) = 4\times2\,T\left(\frac{n}{8}\right) + 4\times\frac{n}{4}*k$ × 4

⋮

$T(1) = K$

n → n/2 → n/4 → n/8 → ... → 1

$T(n) = n*k(a-1) + k$

↑ ~~let say a times~~ n*k, a times aa raha hai

$\Rightarrow \frac{n}{2^a} = 1$

$n = 2^a$

$\log n = \log 2^a$

$\boxed{a = \log n}$

$T(n) = n*k(\log n - 1)$

$T(n) = n * \log n$

$\boxed{T.C = O(n \log n)}$