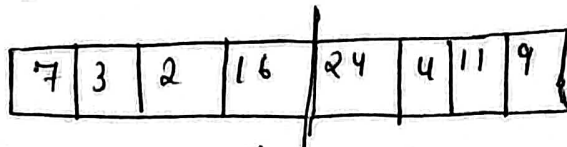


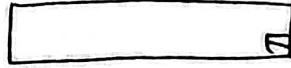
Divide & Conquer

① Merge Sort → based on divide & conquer.

i/p →

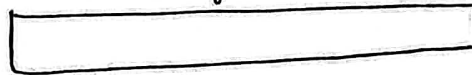


sorted array →



sorted array

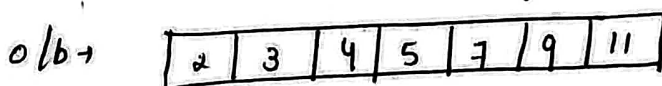
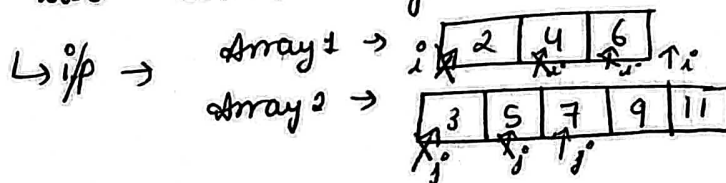
merge



Steps →

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

Q → Merge two sorted arrays.



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 ->

Divide

Conquer

38 | 27 | 43 | 3 | 9 | 20

38 | 27 | 43

3 | 9 | 20

38 | 27

43

3 | 9

20

38

27

43

3

9

20

27 | 38

43

3 | 9

20

27 | 38 | 43

3 | 9 | 20

3 | 9 | 20 | 27 | 38 | 43

→ 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.

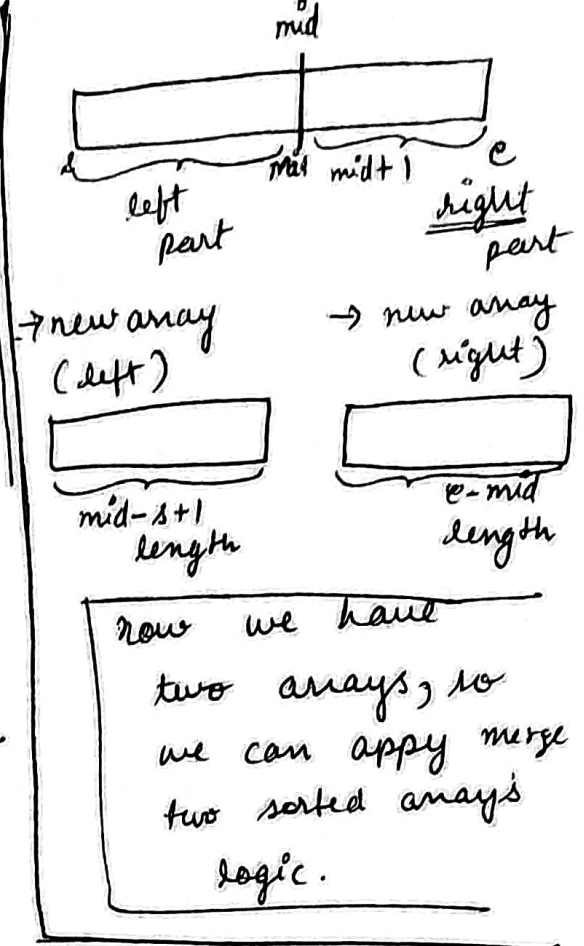
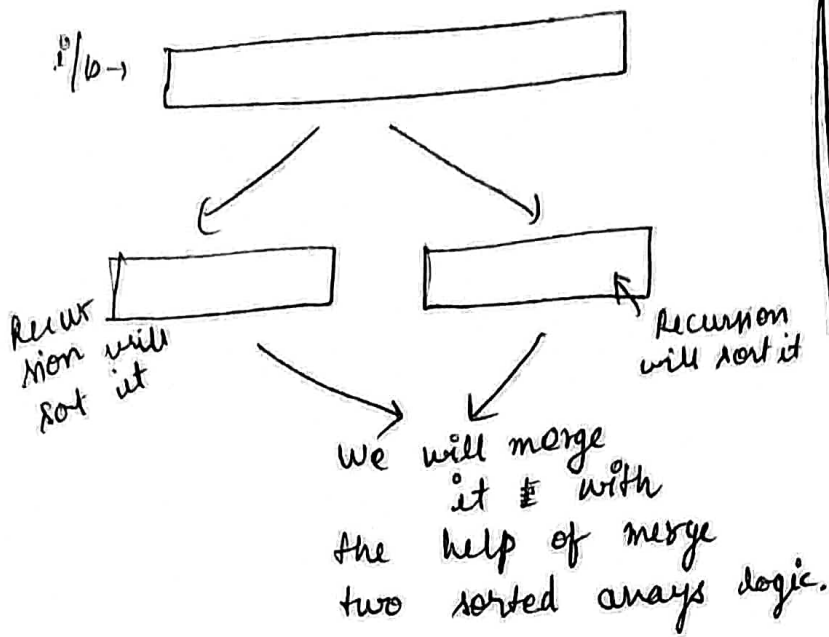
```
int main() {
    int arr[] = {4, 5, 13, 2, 12};
    int n = 5;
    int s = 0, e = n - 1;
    mergeSort(arr, s, e);
}
```

```
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);
mergeSort(arr, mid + 1, e);
merge(arr, s, e);
```

← call for left half and sort.
← call for right half and sort.
← 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 ~~an~~ 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 array;

for (int i = 0; i < len2; i++){

right[i] = arr[k];
k++;

}

// Merge logic →

int leftIndex = 0;

int rightIndex = 0;

int mainArrayIndex = 0;

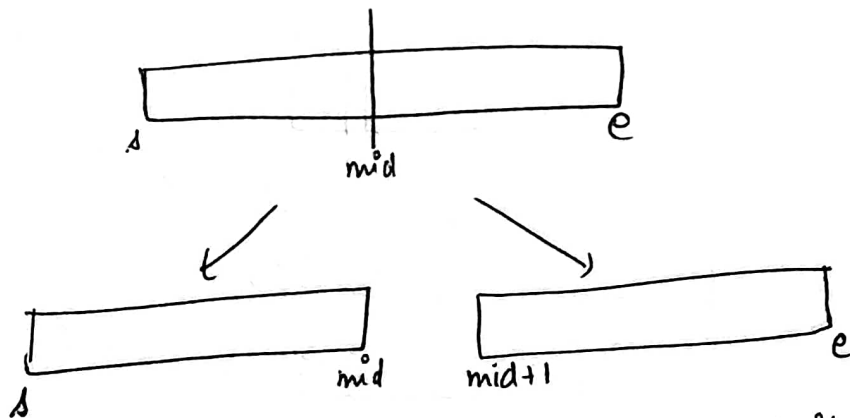
```

while (leftIndex < len1 && rightIndex < len2) {
if (left[leftIndex] < right[rightIndex])
    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, so we will copy them.
while (leftIndex < len1) {
    arr[mainArrayIndex++] = left[leftIndex++];
}

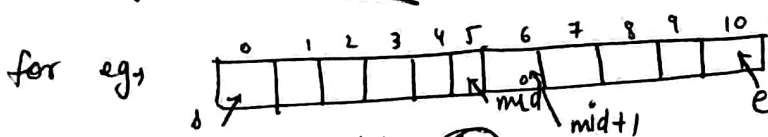
// copy right arrays remaining elements.
while (rightIndex < len2) {
    arr[mainArrayIndex++] = left right[rightIndex++];
}
}

```



$$\text{length} = \text{mid} - s + 1$$

$$\begin{aligned}
 \text{length} &= e - (\text{mid} + 1) + 1 \\
 &= e - \text{mid} - 1 + 1 \\
 &= e - \text{mid}
 \end{aligned}$$



$$\rightarrow \text{length} = 5 - 0 + 1 = \underline{\underline{6}}$$

\rightarrow length of right array

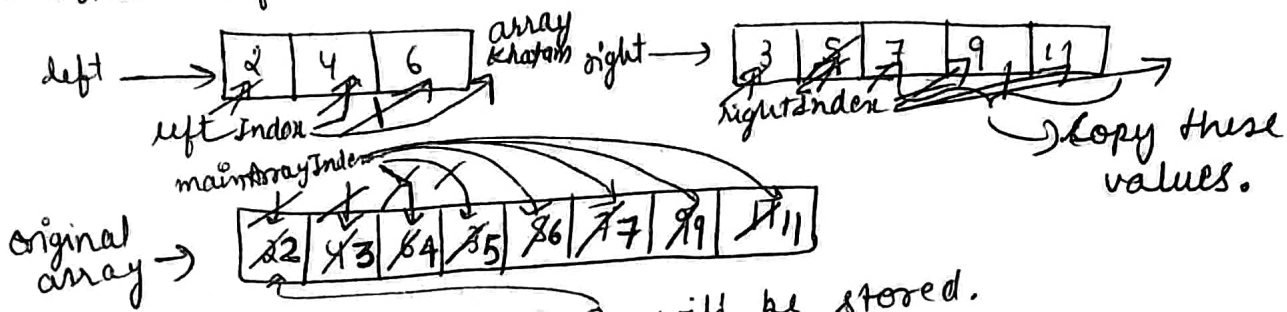
$$\Rightarrow e - e + e - (\text{mid} + 1) + 1$$

$$\Rightarrow e - \text{mid} = 10 - 5 = \underline{\underline{5}}$$

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

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

keyword (returns address)



2 < 3 → T → so 2 will be stored.

4 < 3 → F → so 3

4 < 5 → T → so 4

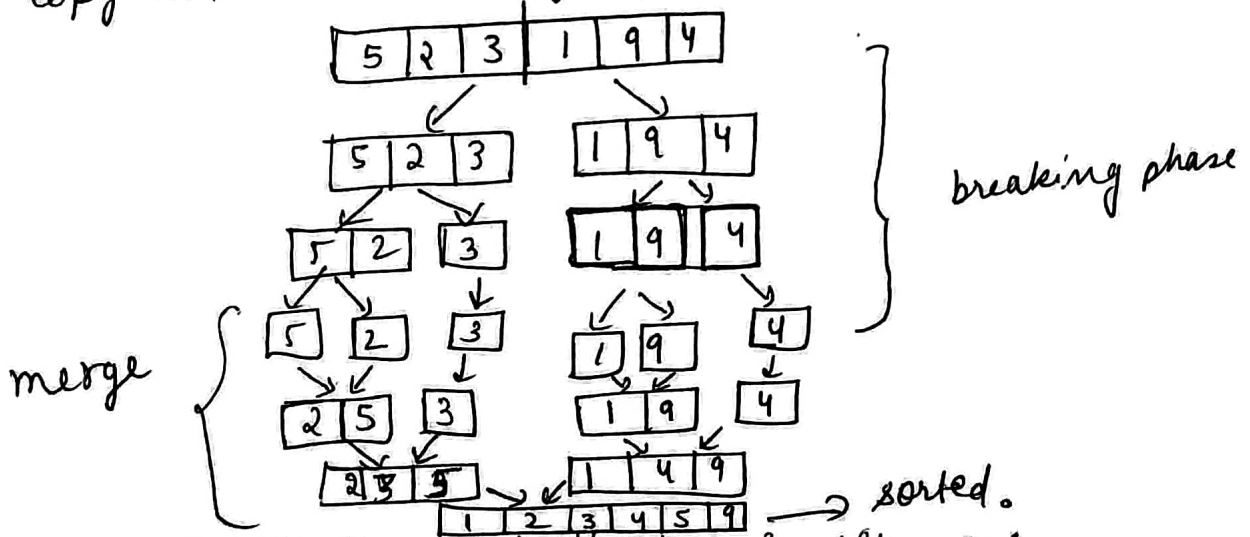
6 < 5 → F → so 5

6 < 7 → T → so 6

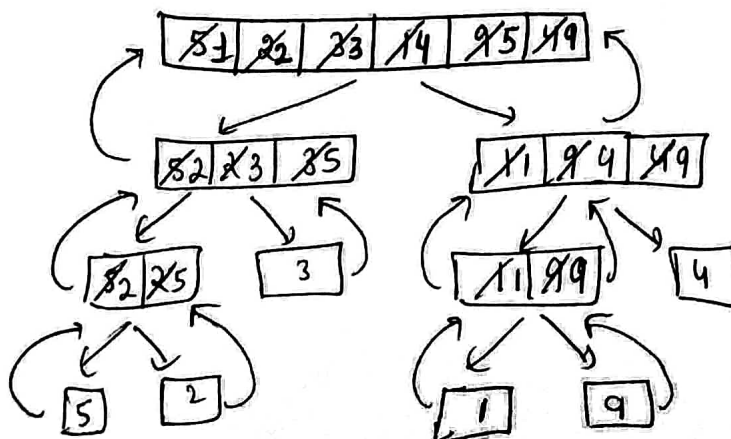
now left array is exhausted, so all the elements of right array will be copied to the array.

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 →



$T.C \rightarrow O(n \log n) \rightarrow$ Find out how?

$S.C \rightarrow O(n) \rightarrow$ Find out how?

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

\rightarrow yes, Inplace Merge sort \rightarrow Explore it.

~~18 Jan~~
H.W - Inversion Count.

① Quick Sort \rightarrow

② Merge Sort T.C \rightarrow

mergesort() {

// BC

// left call

// right call

// merge sort

}

left call right call merge array
↓ ↓ ↓

$$T(n) = k_1 + T(n/2) + T(n/2) + n \times k$$

$$= k_1 + 2T(n/2) + n \times k$$

$$T(n) = 2T(n/2) + n \times k$$

Solving this eqⁿ by ~~elimination~~ ^{substitution} method \rightarrow

$$\begin{array}{l} \left. \begin{array}{l} \text{a times} \\ \uparrow \\ (a-1) \text{ times} \end{array} \right\} \begin{array}{l} T(n) = 2T(n/2) + n \times k \\ 2T(n/2) = 2 \times 2T(n/4) + 2 \times \frac{n}{2} \times k \quad \times 2 \\ 4T(n/4) = 4 \times 2T(n/8) + 4 \times \frac{n}{4} \times k \quad \times 4 \\ \vdots \\ T(1) = k \end{array} \end{array}$$

n
 \downarrow
 $n/2$
 \downarrow
 $n/4$
 \downarrow
 $n/8$
 \downarrow
 1

$$T(n) = \frac{n \times k (a-1)}{2^a} + k$$

\uparrow ~~Be~~ ^{let say a times} $n \times k$, a times aa rha hai

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

$$n = 2^a$$

$$\log n = \log 2^a$$

$$a = \log n$$

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

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

$$T.C = O(n \log n)$$