Divide & Conquer

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

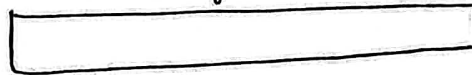i/p →

7	3	2	16	24	4	11	9
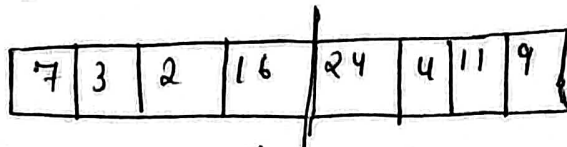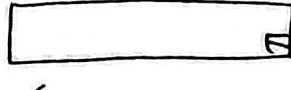
sorted array →

[ ] ← → [ ] ← sorted array

→ merge ←

[ ]

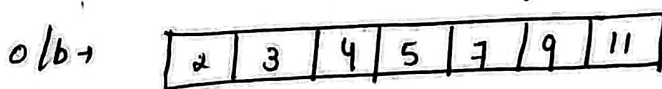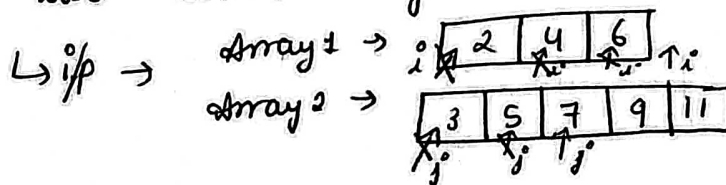**Steps** →

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 → Array1 →

2	4	6

i

Array2 →

3	5	7	9	11

j

o/b →

| 2	3	4	5	7	9	11

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++

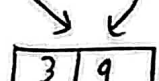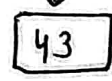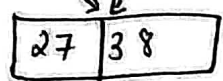if (array1[i] > array2[j])
  store array2[j] and j++

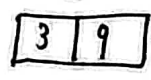→ **Merge Sort →**

$$38 \mid 27 \mid 43 \mid 3 \mid 9 \mid 20$$
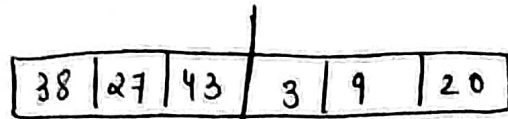
→ break the array into two parts until each part has only one element left. (because one element in any array is already sorted).

Divide

$$38 \mid 27 \mid 43 \qquad 3 \mid 9 \mid 20$$

$$38 \mid 27 \qquad 43 \qquad 3 \mid 9 \qquad 20$$

$$38 \qquad 27 \qquad 43 \qquad 3 \qquad 9 \qquad 20$$

Conquer

$$27 \mid 38 \qquad 43 \qquad 3 \mid 9 \qquad 20$$

→ Merge the array.

$$27 \mid 38 \mid 43 \qquad 3 \mid 9 \mid 20$$

$$3 \mid 9 \mid 20 \mid 27 \mid 38 \mid 43$$
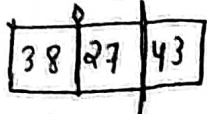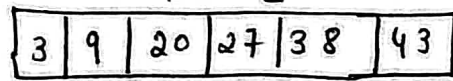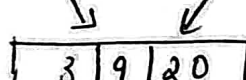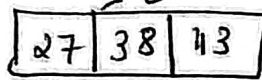
```
int main() {
    int arr[] = {4, 5, 13, 2, 12};
    int n = 5;
    int s = 0, e = n-1;
    mergeSort(arr, s, e);

}

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.
}
```

```cpp
void merge(int *arr, int s, int e){
```

i/p →

Recursion will sort it

Recursion will sort it

We will merge it with the help of merge two sorted arrays logic.

mid

left part

l .... mid mid+1 .... e
right part

→ new array (left)

→ new array (right)

mid-s+1 length

e-mid length

now we have two arrays, so we can apply merge two sorted array's logic.
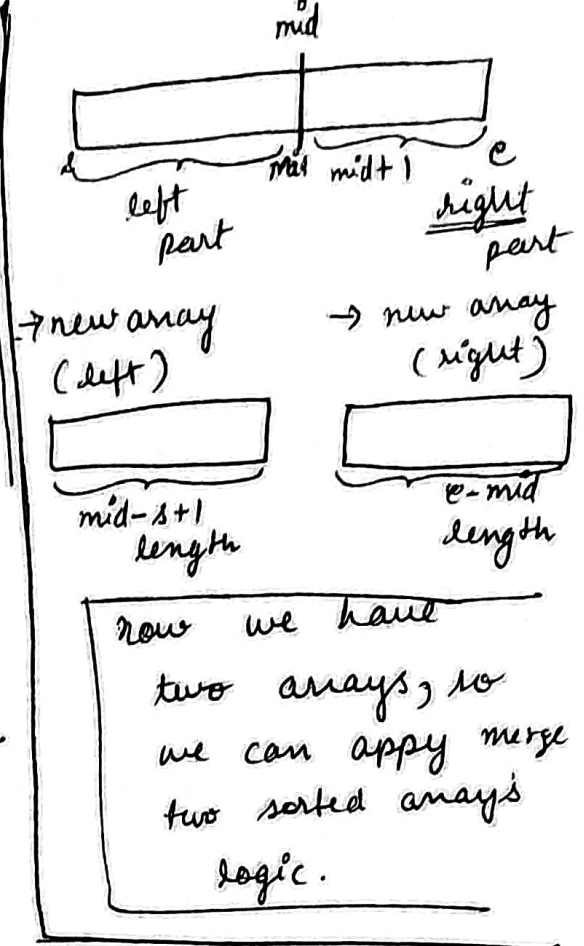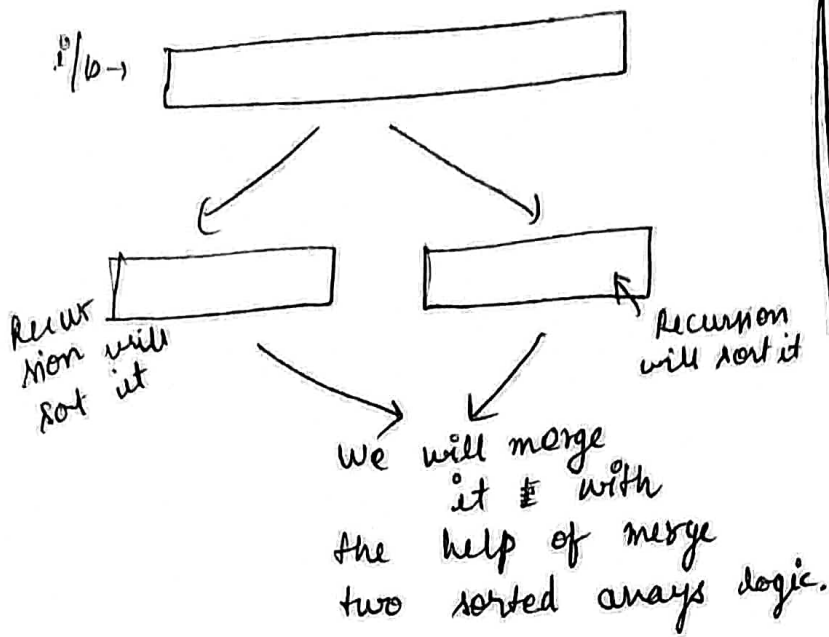
```cpp
void merge(int *arr, int s, int e){
    int mid = s + (e-s)/2;
    int len1 = mid - s + 1;
    int len2 = e - mid;
    int *left = new int[len1];   } → Creating arrays
    int *right = new int[len2];       dynamically of
                                      given lengths.
    // copy values →
    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] < rightIndex- rightIndex
    if ( left [leftIndex] < right [rightIndex]){
        arr [mainArray Index] = left [leftIndex];
        mainArray Index ++;
        leftIndex ++;
    }
    else {
        arr [mainArrayIndex ++] = right [rightIndex ++];
    }
}
```

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

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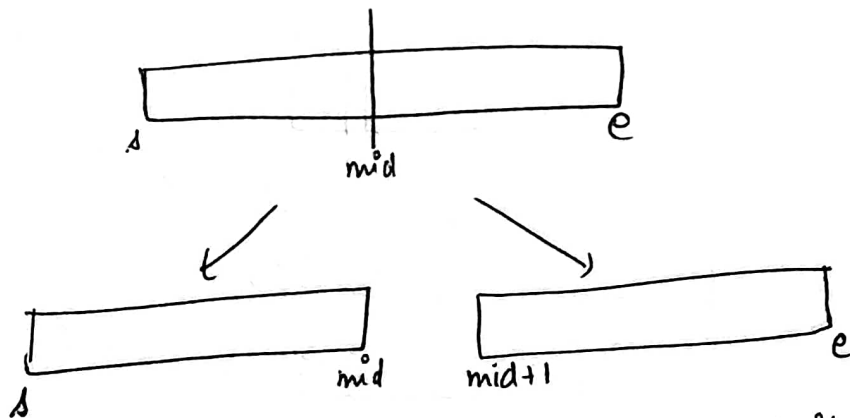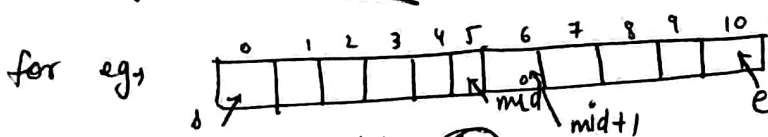

length = mid - s + 1

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

→ length of right array

for eg,

length = 5 - 0 + 1 = ⑥

⇒ e - ei e - (mid + 1) + 1

⇒ e - mid = 10 - 5 = 5

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

0	1	2	3	4	5	6	7	8	9

new → keyword (returns address)

left Array = address

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

left →

2	4	6

↑ leftIndex

array khatam   right →

3	5	7	9	11

rightIndex

→ copy these values.

mainArrayIndex

original array →

2	3	4	5	6	7	9	11
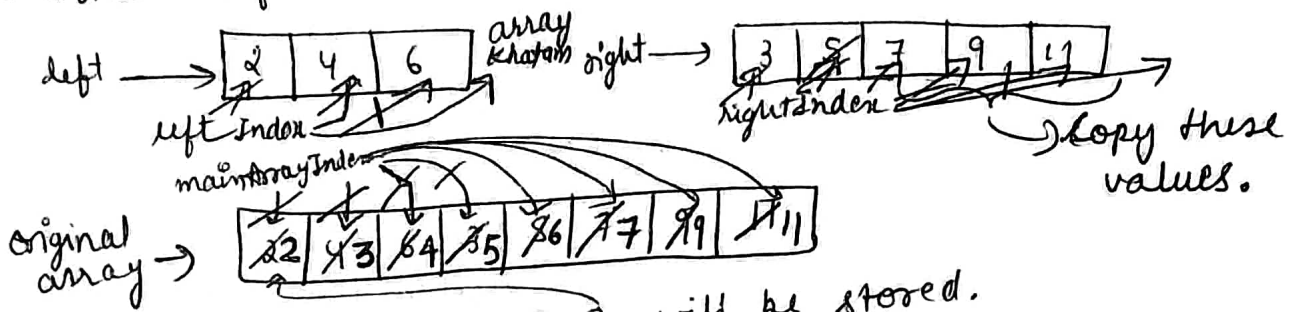
2 < 3 → T → so 2 will be stored.
4 < 3 → F → so 3
4 < 5 → T → so 4
6 < 5 → F → so 5
6 < 7 → T → so 6

now left array is exhauset, 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 untile one of array enhausts.

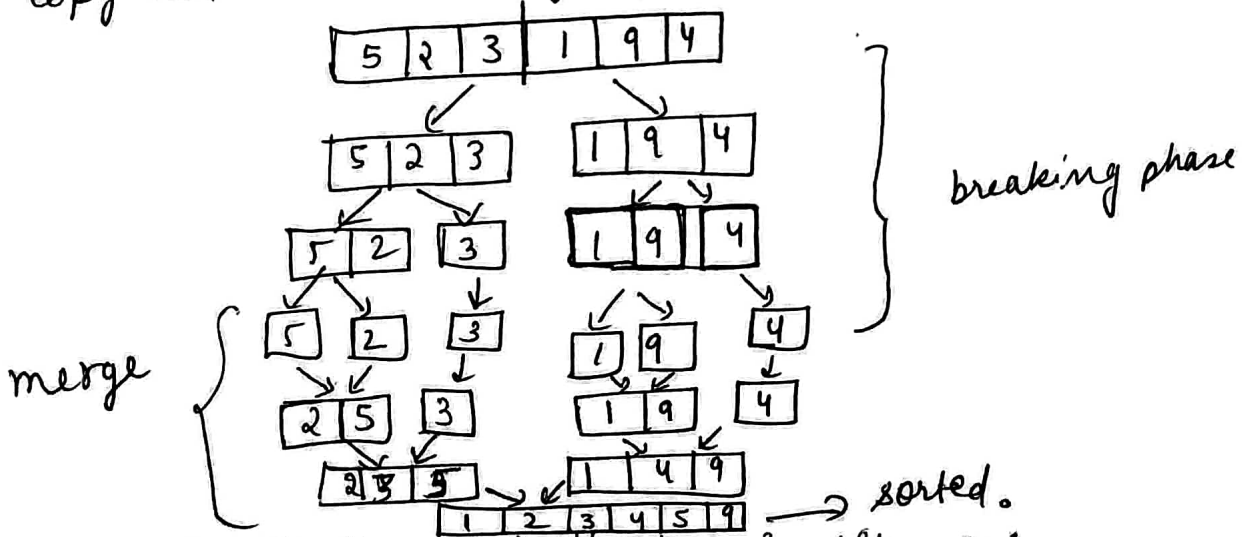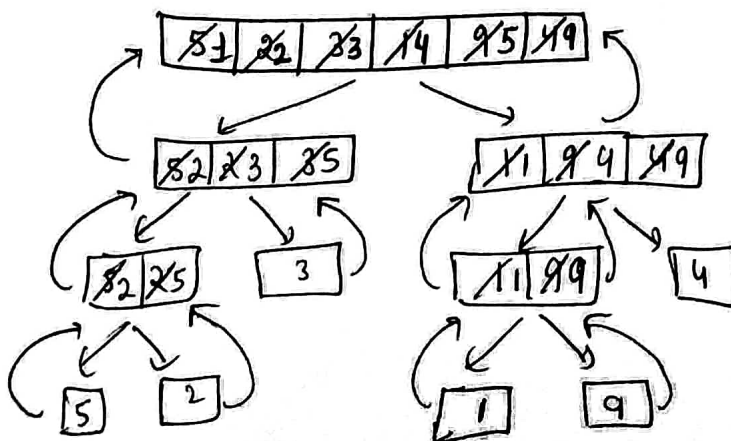② Copy rem. value from left array into main array.

③ Copy rem. " " right " " " " " "

5	2	3	1	9	4

5	2	3

1	9	4

5	2	3

1	9	4

breaking phase

5	2	3

1	9	4

merge

5	2	3

1	9	4

2	5	3

1	9	4

2	3	5

1	4	9

1	2	3	4	5	9

→ sorted.

→ fust for visualization, actually tree is like this →

1	2	3	4	5	9

2	3	5

1	9	4

2	5

3

1	9

4

5

2

1

9

$$T.C \to O(n \log n) \longrightarrow \text{Find out how?}$$

$$S.C \to O(n) \longrightarrow \text{Find out how?}$$

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

↳ Yes, Inplace Merge Sort → Explore it.

H·W - Inversion Count.

Ⓠ **Quick Sort →**

Ⓠ **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 eqⁿ by ~~ilimination~~ substitution method →

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

$$2T\left(\tfrac{n}{2}\right) = 2\times 2 T\left(\tfrac{n}{4}\right) + \cancel{2}\times\tfrac{n}{\cancel{2}}*k \qquad \times 2$$

$$4T\left(\tfrac{n}{4}\right) = 4\times 2 T\left(\tfrac{n}{8}\right) + \cancel{4}\times\tfrac{n}{\cancel{4}}*k \qquad \times 4$$

a times

(a-1) times

$$T(1) = k$$

$$\underline{\hspace{8cm}}$$

$$T(n) = \frac{n*k(a-1) + k}{}$$

↑ n*k, a times aa rha hai

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

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

$$n = 2^a$$

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

$$\boxed{a = \log n}$$

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

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

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

1