18|10|2023

# DIVIDE AND CONQUER
# CLASS 1

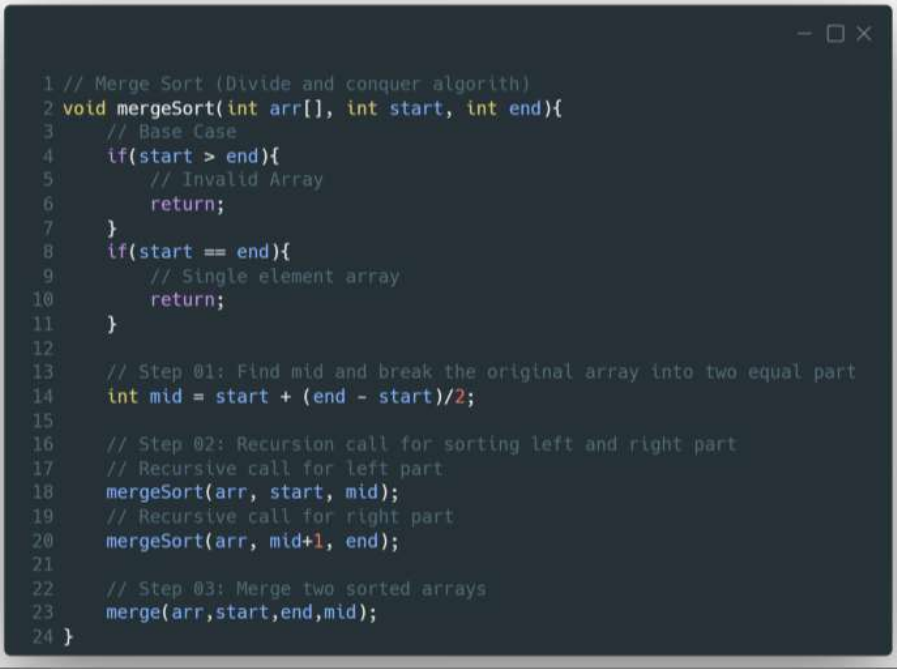
# Merge sort

**Step 01:** *Find mid and break the original array into two equal part*

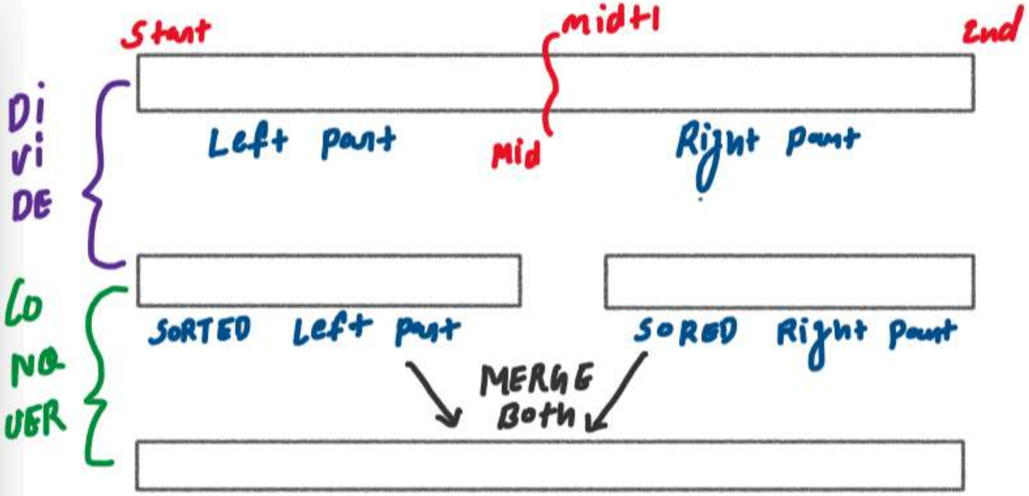
**Left part [start, mid]** *and* **Right part [mid+1, end]**

**Step 02:** *Recursion call for sorting left and right part*

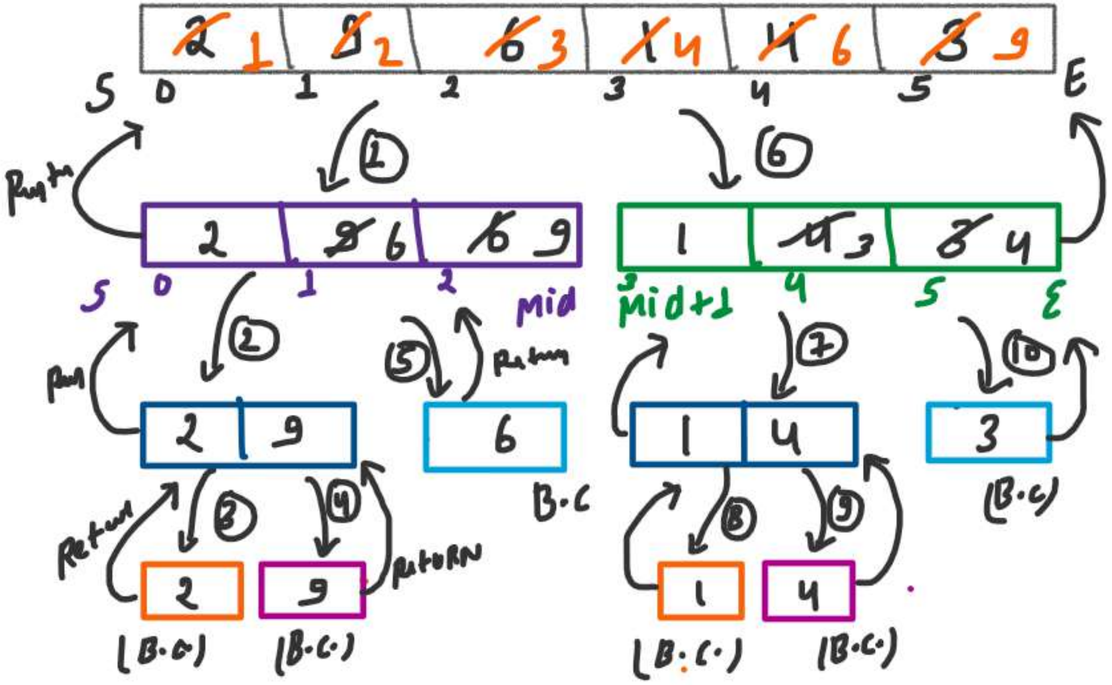
**Step 03:** *Merge two sorted arrays*



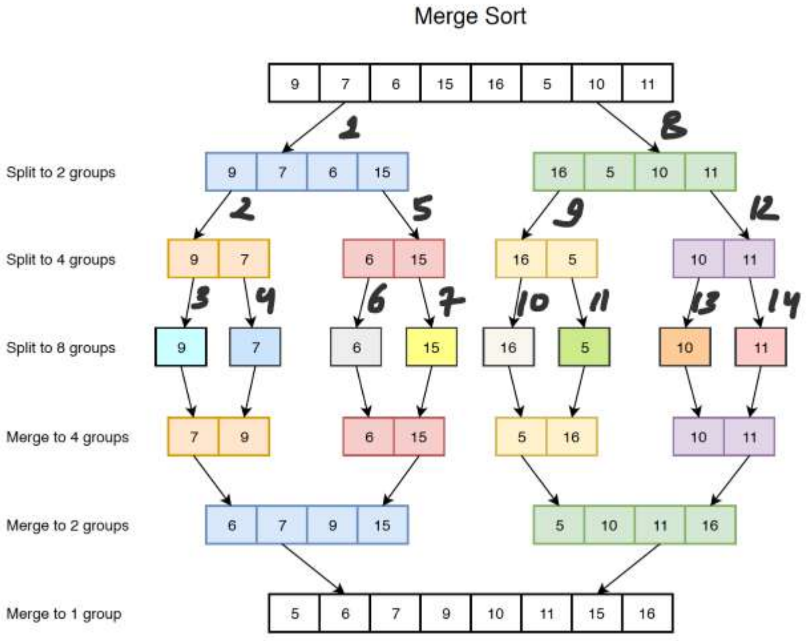
```
// Merge Sort (Divide and conquer algorith)
void mergeSort(int arr[], int start, int end){
    // Base Case
    if(start > end){
        // Invalid Array
        return;
    }
    if(start == end){
        // Single element array
        return;
    }

    // Step 01: Find mid and break the original array into two equal part
    int mid = start + (end - start)/2;

    // Step 02: Recursion call for sorting left and right part
    // Recursive call for left part
    mergeSort(arr, start, mid);
    // Recursive call for right part
    mergeSort(arr, mid+1, end);

    // Step 03: Merge two sorted arrays
    merge(arr,start,end,mid);
}
```



# DRY RUN

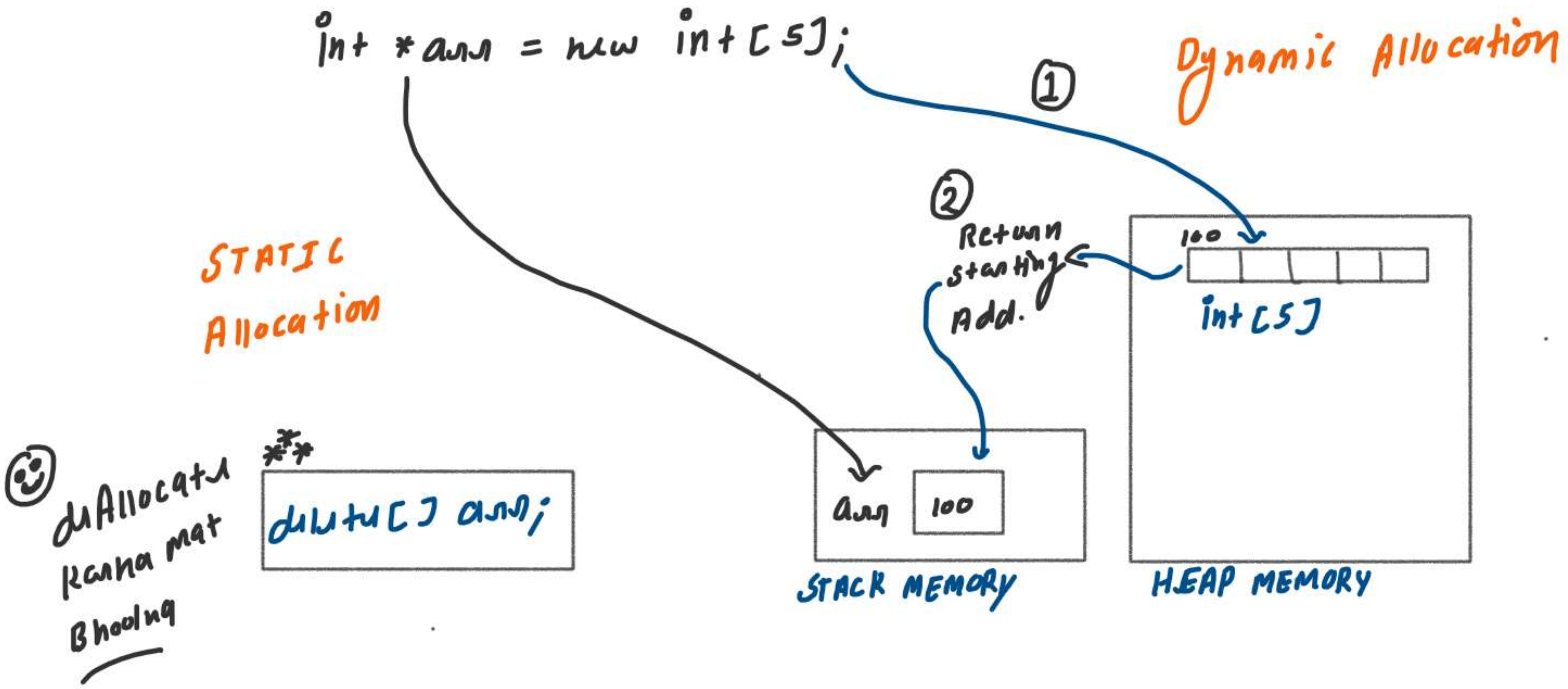


~~2~~ 1
~~9~~ 2
~~6~~ 3
~~1~~ 4
~~4~~ 6
~~3~~ 9
S
0
1
2
3
4
5
E
1
6
Return
2
~~9~~ 6
~~6~~ 9
1
~~4~~ 3
~~3~~ 4
S
0
1
2
mid
3
mid+1
4
5
E
Run
2
5
Return
7
10
2
9
6
B.C
1
4
3
(B.C)
Return
3
4
RETURN
8
9
2
9
1
4
(B.C)
(B.C)
(B.C)
(B.C)



Merge Sort
9 7 6 15 16 5 10 11
1
8
Split to 2 groups
9 7 6 15
16 5 10 11
2
5
9
12
Split to 4 groups
9 7
6 15
16 5
10 11
3
4
6
7
10
11
13
14
Split to 8 groups
9
7
6
15
16
5
10
11
Merge to 4 groups
7 9
6 15
5 16
10 11
Merge to 2 groups
6 7 9 15
5 10 11 16
Merge to 1 group
5 6 7 9 10 11 15 16

# DYNAMIC MEMORY ALLOCATION



```
int *arr = new int[5];
```

```
delete[] arr;
```

# MERGE FUNCTION (STEP:03)

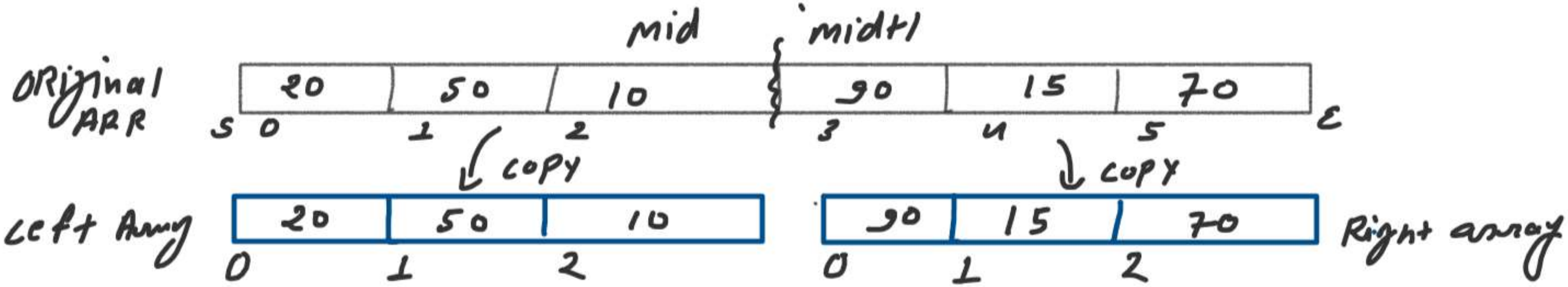
1st step

**Step 01:** *find the length of left and right part array*

**Step 02:** *create left and right part array*

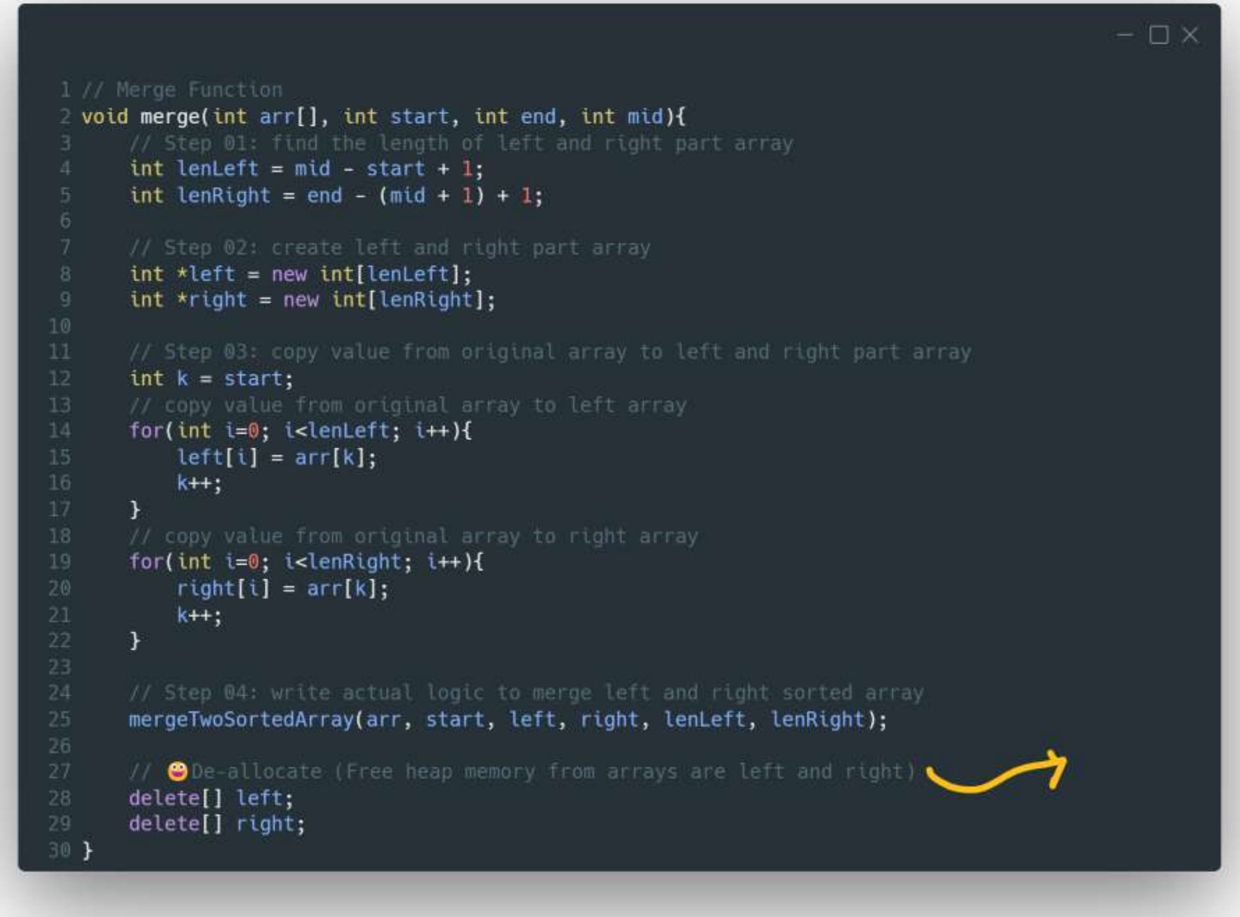
**Step 03:** *copy value from original array to left and right part array*

**Step 04:** *write actual logic to merge left and right sorted array*



$$lenLeft = mid - s + 1$$
$$= 2 - 0 + 1$$
$$= 3$$

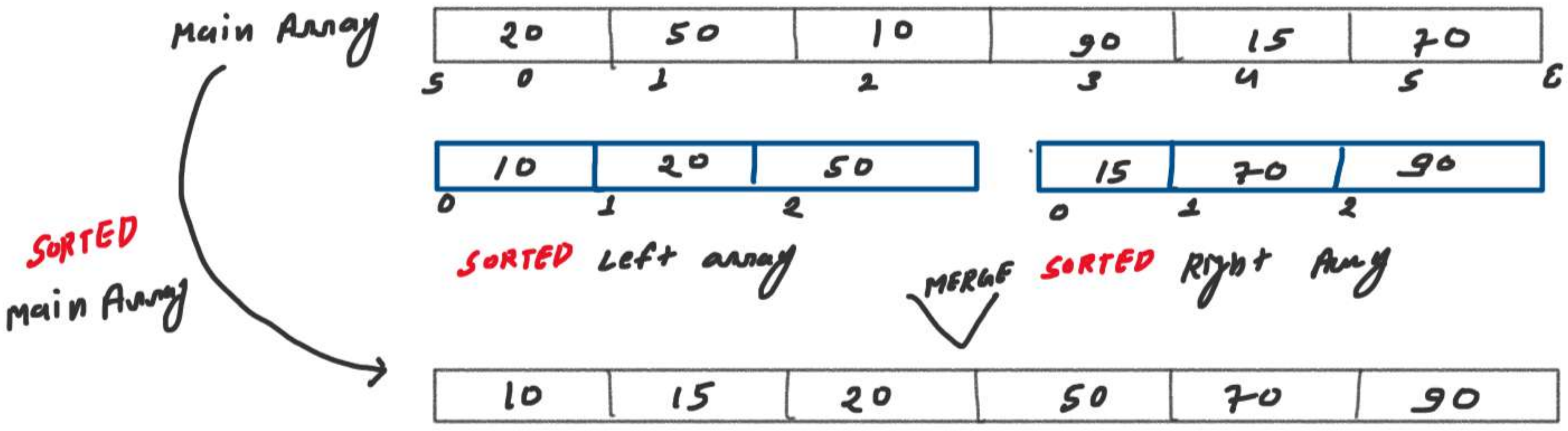
$$lenRight = e - (mid + 1) + 1$$
$$= 5 - (3) + 1$$
$$= 3$$



```cpp
// Merge Function
void merge(int arr[], int start, int end, int mid){
    // Step 01: find the length of left and right part array
    int lenLeft = mid - start + 1;
    int lenRight = end - (mid + 1) + 1;

    // Step 02: create left and right part array
    int *left = new int[lenLeft];
    int *right = new int[lenRight];

    // Step 03: copy value from original array to left and right part array
    int k = start;
    // copy value from original array to left array
    for(int i=0; i<lenLeft; i++){
        left[i] = arr[k];
        k++;
    }
    // copy value from original array to right array
    for(int i=0; i<lenRight; i++){
        right[i] = arr[k];
        k++;
    }

    // Step 04: write actual logic to merge left and right sorted array
    mergeTwoSortedArray(arr, start, left, right, lenLeft, lenRight);

    // 🤪De-allocate (Free heap memory from arrays are left and right)
    delete[] left;
    delete[] right;
}
```

Free Memory Karna mat Bhoolna

## STEP 4 Merge Two SORTED ARRAY



| 20 | 50 | 10 | 90 | 15 | 70 |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

| 10 | 20 | 50 |
|---|---|---|
| 0 | 1 | 2 |

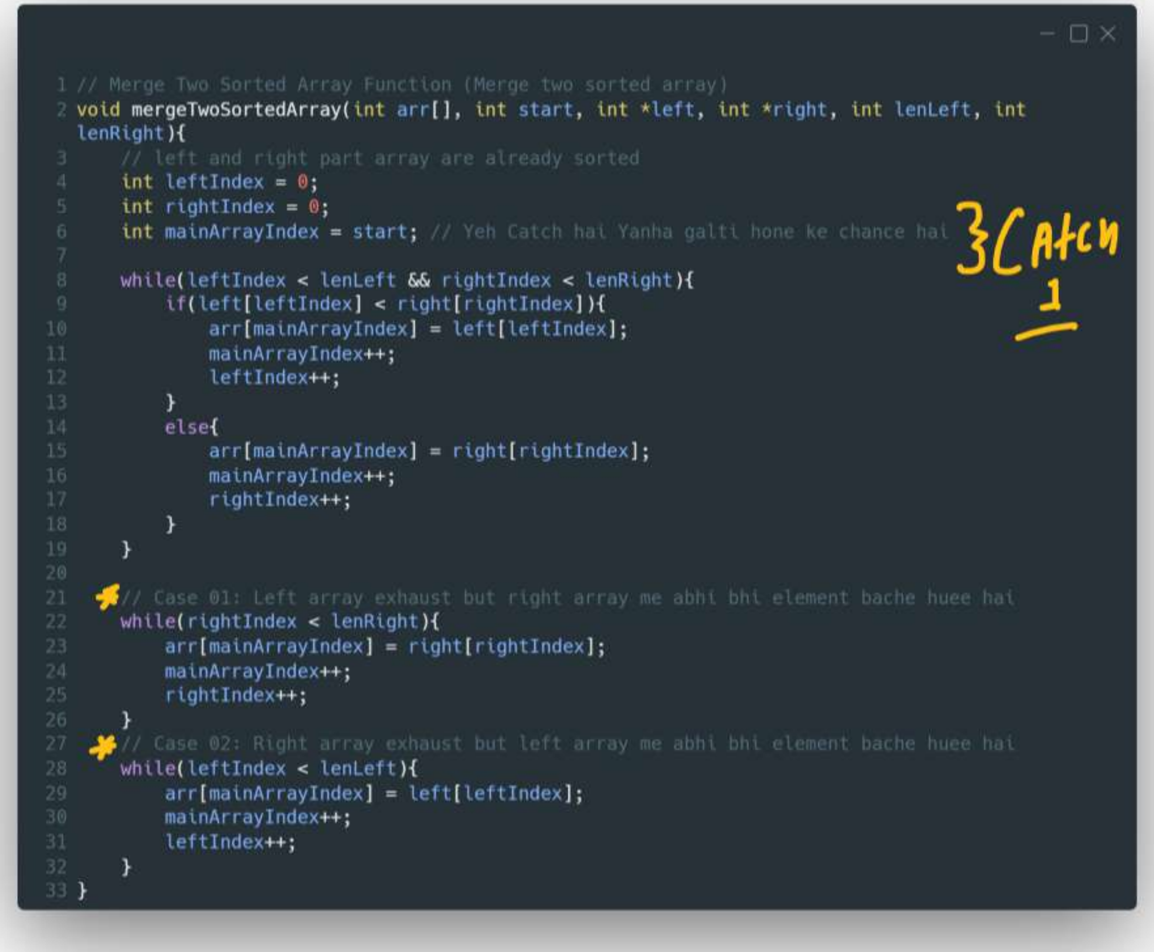
| 15 | 70 | 90 |
|---|---|---|
| 0 | 1 | 2 |

| 10 | 15 | 20 | 50 | 70 | 90 |
|---|---|---|---|---|---|

{ starting Index of all 3 Arrays

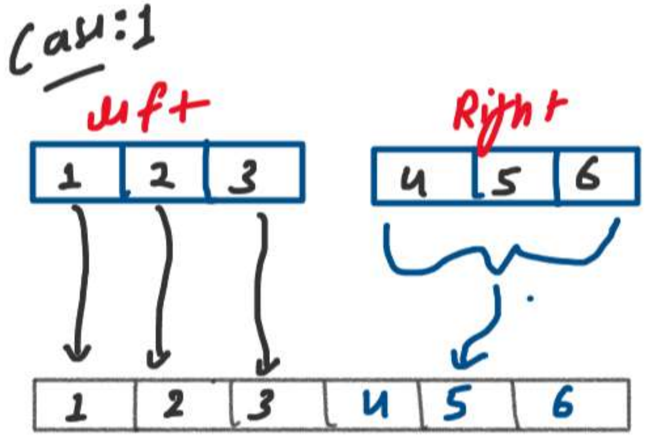
leftIndex = 0
rightIndex = 0
mainArrayIndex = S }

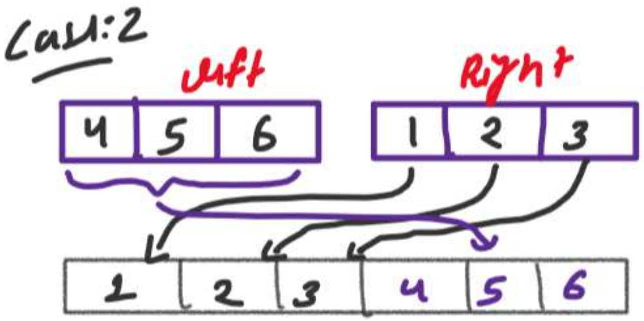
Two Pointers Approach



```
// Merge Two Sorted Array Function (Merge two sorted array)
void mergeTwoSortedArray(int arr[], int start, int *left, int *right, int lenLeft, int
lenRight){
    // left and right part array are already sorted
    int leftIndex = 0;
    int rightIndex = 0;
    int mainArrayIndex = start; // Yeh Catch hai Yanha galti hone ke chance hai

    while(leftIndex < lenLeft && rightIndex < lenRight){
        if(left[leftIndex] < right[rightIndex]){
            arr[mainArrayIndex] = left[leftIndex];
            mainArrayIndex++;
            leftIndex++;
        }
        else{
            arr[mainArrayIndex] = right[rightIndex];
            mainArrayIndex++;
            rightIndex++;
        }
    }

    // Case 01: Left array exhaust but right array me abhi bhi element bache huee hai
    while(rightIndex < lenRight){
        arr[mainArrayIndex] = right[rightIndex];
        mainArrayIndex++;
        rightIndex++;
    }
    // Case 02: Right array exhaust but left array me abhi bhi element bache huee hai
    while(leftIndex < lenLeft){
        arr[mainArrayIndex] = left[leftIndex];
        mainArrayIndex++;
        leftIndex++;
    }
}
```

Catch 1





# Time complexity of merge sort

```
MS( ) {
    Base case   → K1
    MS (left)   → n/2
    MS (Right)  → n/2
    Merge( )    → n*K
}
```

$$T(N) = K_1 + T(n/2) + T(n/2) + n*K$$

$$2^0 * T(N) = 1K_1 + 2\cancel{T(n/2)} + n*K$$

$$2* \cancel{T\left(\frac{N}{2}\right)} = 2K_1 + 4\cancel{T(n/4)} + \cancel{2}\left(\frac{n}{\cancel{2}}*K\right)$$

$$\cancel{4* T\left(\frac{N}{4}\right)} = 4K_1 + 8T\cancel{(n/8)} + \cancel{4}\left(\frac{n}{\cancel{4}}+K\right)$$

$$\cancel{\vdots} \qquad \cancel{\vdots}$$

$$2^{a-1} \cancel{*} T(1) = 2^{a-1} K_1$$

a times

$$T(N) = K\left(1+2+4+..+2^{a-1}\right) + (a-1)(n*K)$$

$$T(N) = K_1(\underbrace{\overset{a}{1}+2+4+8+ \dots +2^{a-1}}_{G.P.}) + (a-1)(n*K)$$

$$\boxed{S_n = a \times \left(\frac{r^n - 1}{r - 1}\right)}$$

Check Binary search

T.C. To understand

$a = \log n$

$$\Rightarrow 1*(2^a - 1) \Rightarrow 2^a \Rightarrow 2^{\log_2 n} \Rightarrow n$$

$$T(N) = \cancel{K_1} n + a*n*\cancel{K}$$ Ignore

$$= n + \log n * n$$

$$= N \log N$$

$$\boxed{O(N\log N)}$$ T.C. of merge sort