Merge Sort :

- Merge sort is a divide and conquer algorithm, it divides the given array into equal parts and then merges the 2 sorted parts.
- There are two functions:

1. merge(): This function merges two halves of the array, assuming both parts are sorted.

2. mergeSort(): This function divides the array into 2 parts (low → mid, mid+1 → high)

// Algorithm & Pseudocode :

- Algorithm:

function 1: mergeSort(arr[], low, high) :

- first divide the array into two parts, divide the arrays range into two halves.

eg: range → 0 to 4,

first half = $\frac{0+4}{2}$: 0 → 2

second half = 3 → 4

// range : low → high : mid = (low+high)/2

1st half : low → mid

2nd half : mid+1 → high

- divide the array by making recursive call:
  - → mergeSort(arr, low, mid) // 1st half
  - → mergeSort(arr, mid+1, high) // 2nd half.

- base case: if (low >= high) return;

// Pseudocode:

```
mergeSort (arr[], low, high) {
  if (low >= high)                    // base case
     return;
  int mid = (low + high)/2;           // find mid

  mergeSort (arr, low, mid);          // left half
  mergeSort (arr, mid+1, high);       // right half.

  merge (arr, low, mid, high);
}
```

- merge (arr[], low, mid, high):

→ use a temp array to store the elements of the two sorted array after merging.

→ use two pointers : one at low (left)

other at mid+1. (right)

→ using while loop (left <= mid && right <= high), select the minimum from the left & right element and insert it in temp array.

→ left-out elements will be copied as it is.

→ transfer elements from temp to range low to high, in original array.

```
merge(arr, low, mid, high){
  temp [];
  left = low;
  right = mid+1;
  while( left <= mid && right <= high) {
```

```
while ( left <= mid && right <= high) {
    if (arr[left] <= arr[high]) {
        temp.add [arr[left]] ;
```

```
        left++;
    }
    else {
        temp.add[arr(right)];
        right++;
    }
}

while (left <= mid) {
    temp.add[arr(left)];
    left++;
}

while (right <= high) {
    temp.add(arr(right));
    right++;
}

for (i = low → high) {
    arr[i] = temp[i-low];
}
}
```

time complexity : O(n log n)
space complexity : O(n)