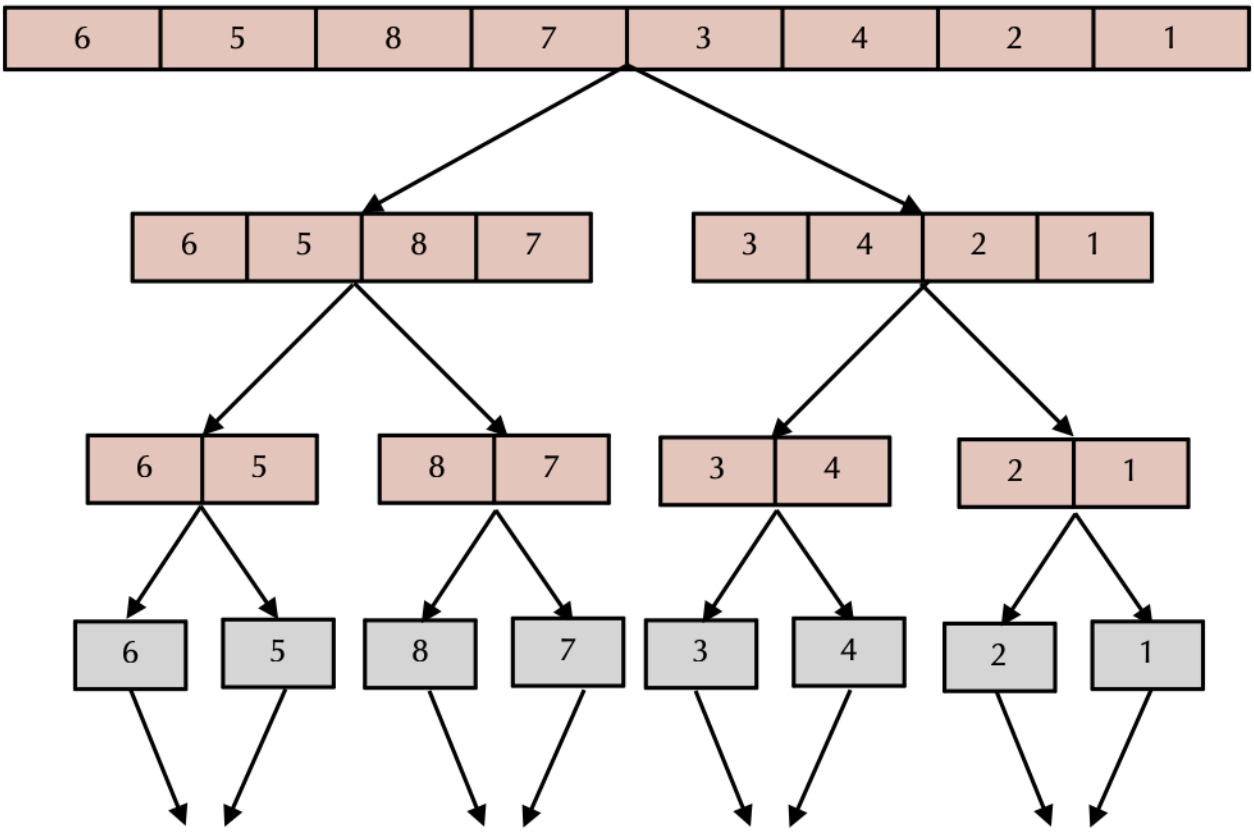
Q3.1) *A* = [6,5,8,7,3,4,2,1]. Perform the Merge Sort algorithm on this array, using a visual illustration of each step to generate the sorted array [1,2,3,4,5,6,7,8]. Show that exactly 12 comparisons are needed to sort this input array A.

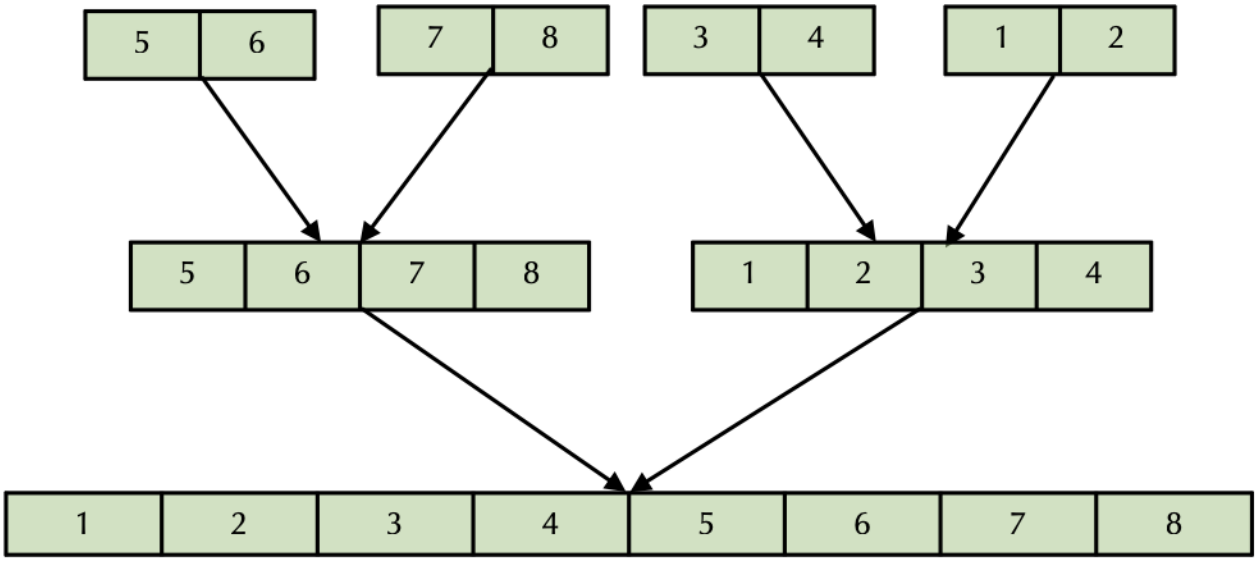
Merge Sort Algorithm uses divide and conquer to sort the array using the following steps-

- It divides the given unsorted array into two halves- left and right sub arrays.
- The sub arrays are divided recursively.
- This division continues until the size of each sub array becomes 1.
- After each sub array contains only a single element, each sub array is sorted trivially.
- The merge procedure explained in the question combines these trivially sorted arrays to produce a final sorted array.

The given array A = [6,5,8,7,3,4,2,1]. Performing merge sort on the following array will out the sorted array A as follows,



Continuing merging the subarrays,



Let's count the number of ***comparisons*** needed to produce the combined (and sorted) merged array.

To get the first green level, we require **4 comparisons**,

1. **(6–5)** - we will compare 6 and 5, and as 5 is smaller than 6, add 5 to merged sorted subarray and then add 6, we get subarray = [5,6].
2. **(8–7)** - we will compare 8 and 7, and as 7 is smaller than 8, add 7 to merged sorted subarray and then add 8, we get subarray = [7,8].
3. **(3–4)** - we will compare 3 and 4, and as 3 is smaller than 4, add 3 to merged sorted subarray and then add 4, we get subarray = [3,4].
4. **(2–1)** - we will compare 2 and 1, and as 1 is smaller than 2, add 1 to merged sorted subarray and then add 2, we get subarray = [1,2].

To get the second green level, we require **4 comparisons**,

1. (**5–7)** - we will compare left-most element of the first subarray 5 with the left-most element of second subarray 7, and as 5 is smaller than 7, add 5 to merged sorted subarray. Subarray = [5]
2. **(6–7)** - Next, we will compare left-most element of the first subarray 6, which is the only element with the left-most element of second subarray

7, and as 6 is smaller than 7, add 6 to merged sorted subarray. Subarray = [5,6]

Now, we can see that the first subarray is empty, and there is nothing left to compare with the second sorted subarray, so we will merge the subarray [7,8] to the sorted subarray, and we get Subarray = [5,6,7,8].

3. **(3–1)** - we will compare left-most element of the third subarray 3 with the left-most element of fourth subarray 1, and as 1 is smaller than 3, add 1 to merged sorted subarray. Subarray = [1]
4. **(3–2)** - Next, we will compare left-most element of the third subarray 3, with the left-most element of fourth subarray 2, and as 2 is smaller than 3, add 2 to merged sorted subarray. Subarray = [1,2]

Now, we can see that the fourth subarray is empty, and there is nothing left to compare with the third sorted subarray, so we will merge the subarray [3,4] to the sorted subarray, and we get Subarray = [1,2,3,4].

To get the third green level, we require **4 comparisons**,

1. (**5–1)** - we will compare left-most element of the first subarray 5 with the left-most element of second subarray 1, and as 1 is smaller than 5, add 1 to merged sorted subarray. Subarray = [1]
2. **(5–2)** - Next, we will compare left-most element of the first subarray 5, with the left-most element of second subarray 2, and as 2 is smaller than 5, add 2 to merged sorted subarray. Subarray = [1,2]
3. **(5–3)** - Next, we will compare left-most element of the first subarray 5 with the left-most element of second subarray 3, and as 3 is smaller than 5, add 3 to merged sorted subarray. Subarray = [1,2,3]
4. **(5–4)** - Next, we will compare left-most element of the first subarray 5 with the left-most element of second subarray 4, and as 4 is smaller than 5, add 4 to merged sorted subarray. Subarray = [1,2,3,4]

Now, we can see that the second subarray is empty, and there is nothing left to compare with the first sorted subarray, so we will merge the first subarray [5,6,7,8] which is already sorted to the merged sorted subarray, and we get Sorted array A = [1,2,3,4,5,6,7,8].

Thus, Merge Sort requires 4 comparisons at each level, 4 + 4 + 4 = 12 total comparisons to sort the array A.

A = [6,5,8,7,3,4,2,1]

Sorted A = [1,2,3,4,5,6,7,8]