**Recurrence Relation:**

T(n) = 1 ; n = 1

T(n) = 2 \* T(n/2) + n ; n > 1

Solution:

T(n/2) = 2 \* T(n/22 ) + n/2

T(n/22) = 2 \* T(n/23) + n/22

T(n/23) = 2 \* T(n / 24) + n/23

By substituting the Above values, we get

T(n) = 2 \* T(n/2) + n

= 2 \* [ 2 \* T(n/22) + n/2] + n 🡺 22  \* T(n/22) + 2n

= 22 \* [ 2 \* T(n/23) + n/22 ] + 2n 🡺 23 \* T( n/23) + 3n

At kth  iteration, T(n) = 2k  \* T(n/2k) + kn

To solve this, we need T(1) = 1.

So, by putting n / 2k = 1 🡺 2k = n 🡺 k = log2 n we will be able to solve this recurrence relation.

= 2log2n \*  T(n/2log2n) + n\*log2n

= n \* T(n ÷ n) + n\* log2n

= n \* T(1) + n \* log2n

∴T(n) = n + n\* log2n = O(n \* log2n)

**Merge Procedure:**

Input: Two Sorted subarrays

Size(Array1) = M && Size(Array2) = N

Output: Sorted Array (Size = M + N )

**Best Case Scenario:**One of the sub-arrays has only one element.

So, the number of comparisons required = min(M, N)

Number of moves required = M + N

∴ Total time taken = # Comparisons + # Moves

= min(M, N) + O(M + N)

= O(M + N)

= O(N) ; N🡪 Total Number of elements = M+N

**Worst Case Scenario:**

Input: Two Sorted Subarrays

Size(Array1) = X && Size(Array2) = Y

Output: Sorted Array (Size = X+Y )

This scenario occurs when elements are compared alternatively from each of the subarrays.

Ex:

Array1: 10, 20, 30, 40

Array2: 11, 21, 31, 41

# Comparisons = O(X+Y)

# Moves = O(X+Y)

∴ Total time taken = # Comparisons + # Moves

= O(X+Y) + O(X+Y)

= O(X+Y); N = X+Y

Total time = O(N)

**Space Complexity:**

Space required to store all the elements +

Number of stack frames used for the recursive calls.

= O(N) + O(log2N)

**Space Complexity Merge Procedure = O(N)**