Recurrence Relation:

$T(n) = 1 ; n = 1$

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

Solution:

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

$T(n/2^2) = 2 * T(n/2^3) + n/2^2$

$T(n/2^3) = 2 * T(n / 2^4) + n/2^3$

By substituting the Above values, we get

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

$= 2 * [2 * T(n/2^2) + n/2] + n \Rightarrow 2^2 * T(n/2^2) + 2n$

$= 2^2 * [2 * T(n/2^3) + n/2^2] + 2n \Rightarrow 2^3 * T(n/2^3) + 3n$

At k^{th} iteration, $T(n) = 2^k * T(n/2^k) + kn$

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

So, by putting $n / 2^k = 1 \Rightarrow 2^k = n \Rightarrow k = \log_2 n$ we will be able to solve this recurrence relation.

$= 2^{\log_2 n} * T(n/2^{\log_2 n}) + n*\log_2 n$

$= n * T(n \div n) + n* \log_2 n$

$= n * T(1) + n * \log_2 n$

$\therefore T(n) = n + n* \log_2 n = O(n * \log_2 n)$

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

$\therefore$ 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

$\boxed{\text{Total time = O(N)}}$

Space Complexity:

Space required to store all the elements +

Number of stack frames used for the recursive calls.

= O(N) + O($\log_2$N)

$\boxed{\textbf{Space Complexity Merge Procedure = O(N)}}$