Time Complexity For Merge Sort Algorithm.

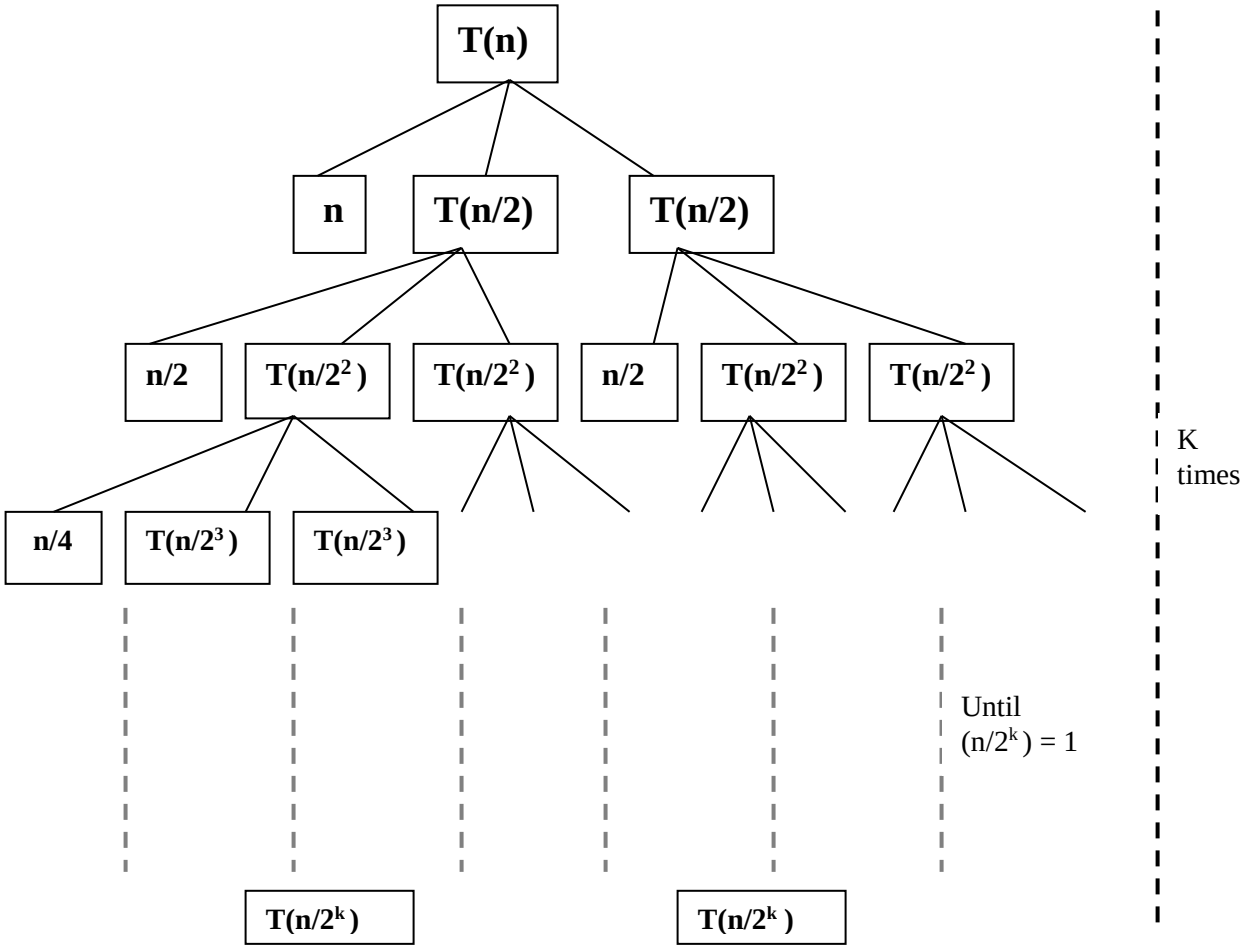
Merge Sort is a divide and conquer algorithm. It works by continuously dividing the array into half, until the size of the array is 1. Then a merge function is called to merge the elements together by sorting them.
In each step, the time complexity for the merge function is O(n) as n elements are merged in each step of the algorithm.

Calculating Time Complexity:

The recurrence relation can be represented as:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + n & n > 1 \end{cases}$$

1. By Recursion Tree Method:



As the algorithm continues until $n/2^k = 1$ (k steps) ,

$$n = 2^k$$
$$\text{i.e. } k = \log_2 n$$

So, the algorithm has log n steps and in each step, n elements are merged. Hence, the total Time take is ***O(n log n).***

2. By Substitution Method:

We know,

$T(n) = 2\,T(n/2) + n$

Or, $T(n) = 2[2T(n/2^2) + n/2] + n$ $\quad [T(n/2) = 2T(n/2^2) + n/2]$

Or. $T(n) = 4T(n/2^2) + n + n$

Or, $T(n) = 4[2T(n/2^3) + n/4] + 2n$ $\quad [T(n/2^2) = 2T(n/2^3) + n/4]$

Or, $T(n) = 2^3T(n/2^3) + 3n$

|

|

Or, $T(n) = 2^k\,T(n/2^k) + kn$

Assume $n/2^k = 1$,

i.e. $k = \log_2 n$

So,

$T(n) = 2^{\log_2 n} * T(1) + n \log_2 n$

i.e. $T(n) = n + n \log_2 n$ $\quad$ [since $T(1) = 1$]

Hence, The time complexity is O(n log n).

Best Case : ***O(n log n)***
Average Case: ***O(n log n)***
Worst Case: ***O(n log n)***

Time complexity of Merge Sort is O(n Log n) in all the 3 cases (worst, average and best) as merge sort always divides the array in two halves and takes linear time to merge two halves. Also, it requires equal amount of additional space as the unsorted array.