Merge sort

How we compute the time complexity of merge sort of n elements.

$T(n)$: Denote the running time of merge sort of n elements.

Divide: The divide step just computes the middle of the subarray, which takes constant time, $O(1)$

Conquer: we recursively solve two subproblems, each of size $n/2$, which contributes $T(n/2) + T(n/2)$ to the running Time

Combine: The merge Procedure on n-element subarray, takes time $O(n)$

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

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

- - - - - - - - - -

- - - - - - - - - - -

on k step (last step)

$$= 2^k T(n/2^k) + kn \longrightarrow (1)$$

$$T(n) = 2T(n/2) + n$$
$$= 2 \cdot \{2 \cdot T(n/4) + n/2\} + n$$
$$= 2^2 T(n/2^2) + n + n$$
$$= 2^2 T(n/4) + 2n .$$
$$= 2^2 \{2 \cdot T(n/2^3) + n/4\} + 2n$$
$$= 2^3 T(n/2^3) + n + 2n$$
$$= 2^3 T(n/2^3) + 3n .$$

So, $n/2^k = 1$

$n = 2^k$

Taking logarithm on bothside of base 2

$$\log_2 n = \log_2 2^k = k \cdot \log_2 2 = k$$

$k = \log_2 n$ (no. of steps = no. of passes)

Put the value of k in equation (1)

$$= 2^{\log_2 n} T\left(n/2^{\log_2 n}\right) + n \log_2 n$$
$$= n \cdot T(1) + n \log_2 n$$
$$= n \cdot 1 + n \cdot \log_2 n$$
$$= n + n \cdot \log_2 n$$

Time Complexity: $O(n \log_2 n)$