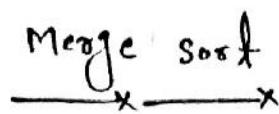


(7)



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 $O(n)$

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

$$\left. \begin{array}{l}
 T(n) = 2T\left(\frac{n}{2}\right) + n \\
 = 2^2 T\left(\frac{n}{2^2}\right) + 2n \\
 = 2^3 T\left(\frac{n}{2^3}\right) + 3n \\
 = 2^4 T\left(\frac{n}{2^4}\right) + 4n \\
 \dots \dots \dots \\
 \text{on } k^{\text{th}} \text{ step (last step)} \\
 \dots = 2^k T\left(\frac{n}{2^k}\right) + kn. \rightarrow \textcircled{1} \\
 \text{So, } \frac{n}{2^k} = 1 \\
 \therefore n = 2^k
 \end{array} \right\} \quad \left. \begin{array}{l}
 T(n) = 2T\left(\frac{n}{2}\right) + n \\
 = 2 \cdot \underbrace{\{2 \cdot T\left(\frac{n}{4}\right) + \frac{n}{2}\}}_{2^2 T\left(\frac{n}{2^2}\right)} + n \\
 = 2^2 T\left(\frac{n}{2^2}\right) + n + n \\
 = 2^2 \underbrace{\{2 \cdot T\left(\frac{n}{8}\right) + \frac{n}{4}\}}_{2^3 T\left(\frac{n}{2^3}\right)} + 2n \\
 = 2^3 T\left(\frac{n}{2^3}\right) + n + 2n \\
 = 2^3 T\left(\frac{n}{2^3}\right) + 3n.
 \end{array} \right.$$

Taking logarithm on L.H.S. side
of base 2

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

$$k = \log_2 n \quad (\text{no. of steps} = \text{no. of passes})$$

Put the value of k in equation (1)

$$\begin{aligned}
 &= 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + n \log_2 n \\
 &= n \cdot T(1) + n \log_2 n
 \end{aligned}$$

$$= n \cdot 1 + n \cdot \log_2 n$$

$$= n + n \cdot \log_2 n$$

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