

① If $t_1(n) \in o(g_1(n))$ and $t_2(n) \in o(g_2(n))$, then $t_1(n) + t_2(n) \in o(\max\{g_1(n), g_2(n)\})$. Prove the assertions.

② We need to show that $t_1(n) + t_2(n) \in o(\max\{g_1(n), g_2(n)\})$. This means there exists a positive constant c and n_0 such that $t_1(n) + t_2(n) \leq c$

$$t_1(n) \leq c_1 g_1(n) \text{ for all } n \geq n_1$$

$$t_2(n) \leq c_2 g_2(n) \text{ for all } n \geq n_2$$

$$\text{Let } n_0 = \max\{n_1, n_2\} \text{ for all } n \geq n_0$$

consider $t_1(n) + t_2(n)$ for all $n \geq n_0$

$$t_1(n) + t_2(n) \leq c_1 g_1(n) + c_2 g_2(n)$$

We need to relate $g_1(n)$ and $g_2(n)$ to $\max\{g_1(n), g_2(n)\}$:

$$g_1(n) \leq \max\{g_1(n), g_2(n)\} \text{ and}$$

$$g_2(n) \leq \max\{g_1(n), g_2(n)\}$$

Thus,

$$c_1 g_1(n) \leq c_1 \max\{g_1(n), g_2(n)\}$$

$$c_2 g_2(n) \leq c_2 \max\{g_1(n), g_2(n)\}$$

$$c_1 g_1(n) + c_2 g_2(n) \leq (c_1 + c_2) \max\{g_1(n), g_2(n)\}$$

By the definition of big-o notation

$$t_1(n) \in o(g_1(n)) \text{ and } t_2(n) \in o(g_2(n)), \text{ then } t_1(n) + t_2(n) \in o(\max\{g_1(n), g_2(n)\})$$

Thus, the assertion is proved

③ Find the time complexity of the recurrence equation.

Let us consider such that recurrence for Merge sort

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

By using master Theorem

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$, $b > 1$ and $f(n)$ is positive function

ex:-

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

$$a = 2, b = 2, f(n) = n$$

by comparing of $f(n)$ with $n^{\log_b a}$

$$\log_b a = \log_2 2 = 1$$

compare $f(n)$ with $n^{\log_b a}$ $f(n) = n$

$$n^{\log_b a} = n^1 = n$$

$$* f(n) = O(n^{\log_b a}) \text{ then } T(n) = O(n^{\log_b a} \log n)$$

In our case.

$$\log_b a = 1$$

$$T(n) = O(n^1 \log n) = O(n \log n)$$

$$\therefore T(n) = 2T(n/2) + n \text{ is } O(n \log n)$$

$$③ \quad T(n) = \begin{cases} 2T(n/2) + 1 & \text{if } n > 1 \\ 1 & \text{otherwise} \end{cases}$$

By applying of Master theorem

$$T(n) = aT(n/b) + f(n) \quad \text{where } a \geq 1, b > 1$$

$$T(n) = 2T(n/2) + 1$$

here $a=2, b=2, f(n)=1$

By comparison of $f(n)$ and $n^{\log_b a}$

If $f(n) = O(n^c)$ where $c < \log_b a$ then $T(n) = O(n^{\log_b a})$

If $f(n) = O(n^{\log_b a})$, then $T(n) = O(n^{\log_b a} \log n)$

If $f(n) = \Omega(n^c)$ where $c > \log_b a$ then $T(n) = O(f(n))$

lets calculate $\log_b a$

$$\log_b a = \log_2 2 = 1$$

$$f(n) = 1$$

$$n^{\log_b a} = n^1 = n$$

$f(n) = O(n)$ with $c < \log_b a \rightarrow i$

In this case $c=0$ and $\log_b a = 1$

$c < 1$, so $T(n) = O(n^{\log_b a}) = O(n^1) = O(n)$

Time complexity of Merge Sort Relation

$$T(n) = 2T(n/2) + 1 \text{ is } O(n)$$

$$T(n) = \begin{cases} 2T(n-1) & \text{if } n > 0 \\ 1 & \text{otherwise} \end{cases}$$

Here, where $n=0$, $T(0)=1$

Recurrence relation analysis

For $n > 0$:

$$T(n) = 2T(n-1)$$

$$T(n) = 2T(n-1)$$

$$T(n-1) = 2T(n-2)$$

$$T(n-2) = 2T(n-3)$$

$$T(1) = 2T(0)$$

From this pattern

$$T(n) = 2 \cdot 2 \cdot 2 \cdot \dots \cdot 2 \cdot T(0) = 2^n T(0)$$

Since $T(0)=1$ we have

$$T(n) = 2^n$$

The recurrence relation is

$$T(n) = 2T(n-1) \text{ for } n > 0 \text{ and } T(0) = 1 \text{ is } T(n) = 2^n$$

⑤ Big O Notation. Show that $f(n) = n^2 + 3n + 5$ is $O(n^2)$

⑤ $f(n) = O(g(n))$ means $c > 0$ and $n_0 \geq 0$

$$f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0$$

Given is $f(n) = n^2 + 3n + 5$

$c > 0$, $n_0 \geq 0$ such that $f(n) \leq c \cdot n^2$

$$f(n) = n^2 + 3n + 5$$

let's choose $c = 2$

$$f(n) \leq 2 \cdot n^2$$

$$f(n) = n^2 + 3n + 5 \leq n^2 + 3n^2 + 5n^2 \\ = 9n^2$$

so, $c = 9$, $n_0 = 1$ $f(n) \leq 9n^2$ for all $n \geq 1$

$$f(n) = n^2 + 3n + 5 \text{ is } O(n^2)$$