# Time Complexity of an Algorithm

Running time of an algorithm or a program is usually characterized as a function of the input size.

If the input size is $n$, the time complexity is often described in terms of the "order" notation as $O(g(n))$ where $g(n)$ is a positive function of the input size $n$.

## Examples

- We say the time complexity of binary search is $O(\log n)$ where $n$ is the number of elements in the input sorted array.

- Given a list of $n$ elements to be sorted, the complexity of the mergesort algorithm is $O(n \log n)$.

- Given two square matrices with dimension $n$, the computational complexity of adding the two matrices is $O(n^2)$.

The following complexity characterizations are commonly used:

| | |
|---|---|
| $O(1)$ | constant time |
| $O(\log n)$ | Logarithmic time |
| $O(n)$ | Linear time |
| $O(n \log n)$ | Linear times Log time |
| $O(n^2)$ | Quadratic time |
| $O(n^3)$ | cubic time |
| $O(n^k)$ for $k = 1, 2, \ldots$ | polynomial time |
| $O(2^n)$ | Exponential time |
| $O(3^n)$ | Exponential time |
| ⋮ | ⋮ |

The "big oh" notation describes asymptotic upper bound on running time.

asymptotic refers to $n \to \infty$

**Definition**

We say $f(n)$ is $O(g(n))$ if there exists a real constant $c$ (finite) and a finite positive integer $n_0$ such that

$$f(n) \leq c\, g(n) \quad \forall n \geq n_0$$

**Example 1:** $f(n) = 2n^2 + 10$

$g(n) = n^2$

Suppose we choose $c = 3$

| $n$ | $f(n) = 2n^2+10$ | $c\,g(n) = 3n^2$ |
|---|---|---|
| 1 | 12 | 3 |
| 2 | 18 | 12 |
| 3 | 28 | 27 |
| 4 | 42 | 48 |
| 5 | 60 | 75 |
| ⋮ | ⋮ | ⋮ |

Thus we have found a constant $c = 3$ and a positive integer $n_0 = 4$ such that

$$f(n) \leq c\, g(n) \quad \forall n \geq n_0$$

So, $2n^2 + 10$ is $O(n^2)$

If we choose $c = 4$, then $n_0 = 3$ will serve the purpose.

Example : $f(n) = 100n$

$g(n) = n^2$

Let us choose $C = 10$ and find out an appropriate value for $n_0$.

| $n$ | $100n$ | $10n^2$ |
|---|---|---|
| 1 | 100 | 10 |
| 2 | 200 | 40 |
| 3 | 300 | 90 |
| 4 | 400 | 160 |
| 5 | 500 | 250 |
| 6 | 600 | 360 |
| 7 | 700 | 490 |
| 8 | 800 | 640 |
| 9 | 900 | 810 |
| 10 | 1000 | 1000 |
| 11 | 1100 | 1210 |
| ⋮ | ⋮ | ⋮ |

Thus the value of $n_0 = 11$. If we initially chose $C = 20$, the value of $n_0$ would be 6.

Therefore $100n$ is $O(n^2)$. In fact, $100n$ is also $O(n)$ with a choice of $C > 100$.

To show that Binary Search is $O(\log n)$.

We have already seen that

$$T(1) = 1$$
$$T(n) = 1 + T\left(\frac{n}{2}\right)$$

where $T(n)$ is the number of "probes" made by binary search in the worst-case on a sorted array of $n$ elements.
By unfolding the above recursive relationship, assuming $n = 2^k$, we get

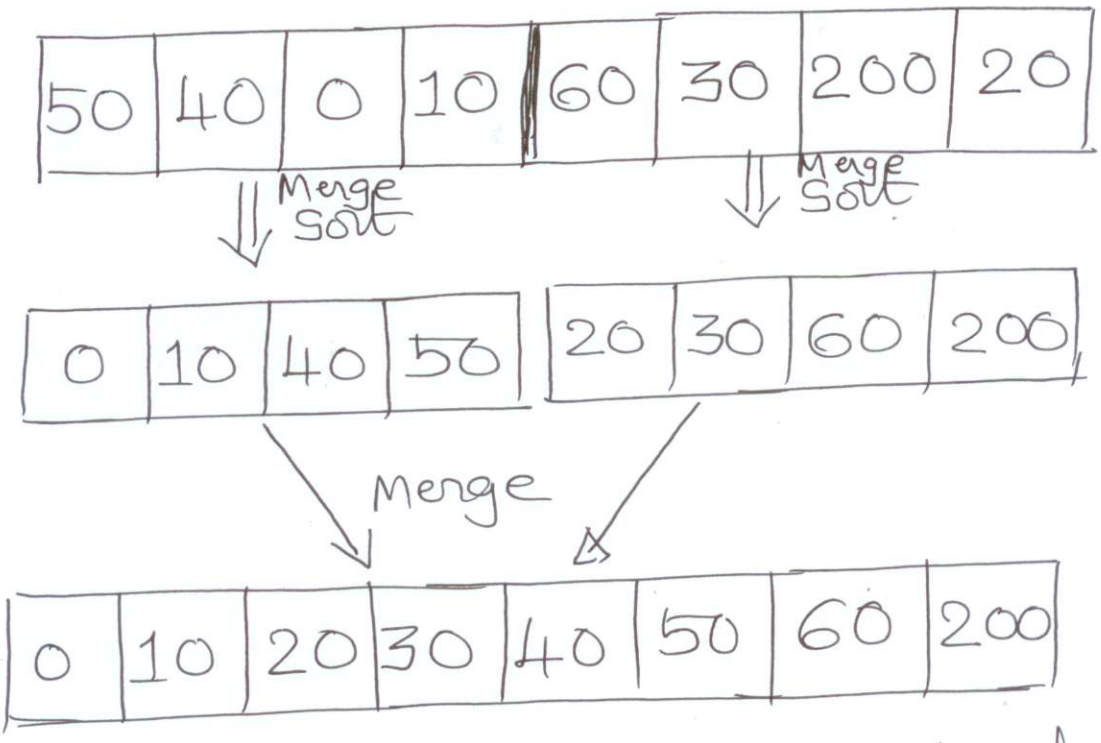
$$\begin{aligned} T(2^k) &= 1 + T(2^{k-1}) \\ &= 1 + (1 + T(2^{k-2})) \\ &= 2 + T(2^{k-2}) \\ &= \cdots \\ &= k + T(1) \\ &= k + 1 \\ &= \log_2 n + 1 \end{aligned}$$

If $n$ is not a power of 2, then we take $k$ such that

$$2^k < n \quad \text{and} \quad 2^{k+1} > n$$

Example: To show that "MergeSort" algorithm is $O(n \log n)$.

MergeSort is a divide-and-conquer algorithm for sorting an array of n elements.




Details of the algorithm will be described subsequently in the course.

The following recursive relation describes the computational complexity:

$$T(1) = 1$$
$$T(n) = 2T\left(\frac{n}{2}\right) + Cn$$

where $C$ is some constant

Suppose $n = 2^k$. Then

$$\begin{aligned} T(n) &= 2T(2^{k-1}) + c \cdot 2^k \\ &= 2\left\{2T(2^{k-2}) + c \cdot 2^{k-1}\right\} + c \cdot 2^k \\ &= 2^2 T(2^{k-2}) + 2c \cdot 2^k \\ &= 2^3 T(2^{k-3}) + 3c \cdot 2^k \\ &= \cdots \\ &= 2^k T(1) + kc \cdot 2^k \\ &= n + c\, n \log n \end{aligned}$$

$T(n)$ can be shown to be $O(n \log n)$.
Left as an exercise.

---

Such relations are called recurrence relations.

Example: Let us show that the function $f(n) = n^3$ is not $O(n^2)$.

Assume to the contrary. That is, assume that $f(n) = n^3$ is $O(n^2)$. This implies there exists a finite real constant $C$ and a finite positive integer ~~such~~ $n_0$ such that

$$n^3 \leq C n^2 \quad \forall n \geq n_0 \quad \cdots (1)$$

This implies that

$$n \leq C \quad \forall n \geq n_0$$

which in turn implies that

$$C \geq n \quad \forall n \geq n_0$$

The above clearly means that $C$ cannot be finite which contradicts our assumption.

Another way of arriving at the contradiction would be: Equation (1) implies that

$$n^3 \leq C n^2 \text{ only for } n \leq C$$

However we want this to be satisfied $\forall n \geq n_0$ which is impossible since $C$ is finite