

Question 1

Little Oh, o

We use o -notation to denote an upper bound that is not asymptotically tight. Formally, a function $f(n)$ is in the set $o(g(n))$ if, for any positive constant c , there exists a value n_0 such that:

$$0 \leq f(n) < c \cdot g(n) \quad \text{for all } n > n_0$$

In simpler terms, $f(n)$ grows much faster than $g(n)$.

Example: $f(n) = 4n^3 + 10n^2 + 5n$ then $g(n) = n^4$

Big Omega, Ω

The notation $\Omega(n)$ is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity. Formally, a function $f(n)$ is in the set $\Omega(g(n))$ if, for any positive constant c , and for sufficiently large n , there exists a constant n_0 such that:

$$0 \leq c \cdot g(n) \leq f(n) \quad \text{for all } n > n_0$$

In simpler terms, $f(n)$ grows at least as fast as $g(n)$.

Example: $f(n) = 4n^3 + 10n^2 + 5n$ then $g(n) = n^3$

Little Omega, ω

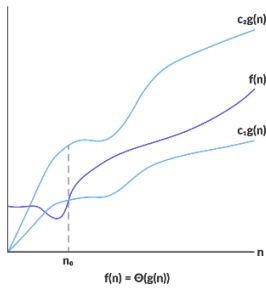
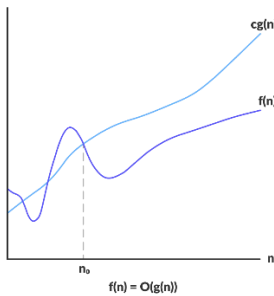
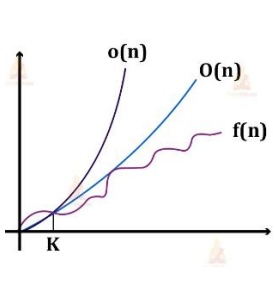
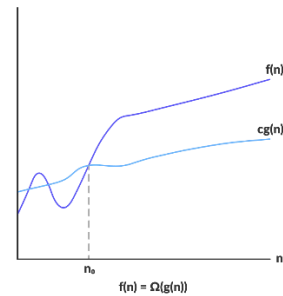
We use ω -notation to denote a lower bound that is not asymptotically tight. Formally, a function $f(n)$ is in the set $\omega(g(n))$ if, for any positive constant c , and for sufficiently large n , there exists a constant n_0 such that:

$$0 \leq c \cdot g(n) < f(n) \quad \text{for all } n > n_0$$

In simpler terms, $f(n)$ grows much faster than $g(n)$.

Example: $f(n) = 4n^3 + 10n^2 + 5n$ then $g(n) = n^2$

Question 2

θ (Theta)	O (Big O)	o (Little Oh)	ω (Little Omega)
$\theta(g(n))$ represents a tight bound. It combines both the upper and lower bounds.	$O(g(n))$ represents an upper bound.	$o(g(n))$ represents a strict upper bound.	$\omega(g(n))$ represents a strict lower bound.
If $f(n)$ is in $\theta(g(n))$, then $f(n)$ grows at the same rate as $g(n)$.	If $f(n)$ is in $O(g(n))$, then $f(n)$ grows at most as fast as $g(n)$.	If $f(n)$ is in $o(g(n))$, then $f(n)$ grows strictly slower than $g(n)$.	If $f(n)$ is in $\omega(g(n))$, then $f(n)$ grows strictly faster than $g(n)$.
Formal way to express both the lower bound and the upper bound of an algorithm's running time.	Formal way to express the lower bound of an algorithm's running time.	We use o -notation to denote an upper bound that is not asymptotically tight.	We use ω -notation to denote a lower bound that is not asymptotically tight.
Describes average case scenario	Specifically describes worst case scenario	Describes worst case scenario	Describes best case scenario
 <p>$f(n) = \Theta(g(n))$</p>	 <p>$f(n) = O(g(n))$</p>	 <p>$f(n) = o(n)$</p>	 <p>$f(n) = \Omega(g(n))$</p>

Question 3

1. In the worst case, this version would still have a time complexity of $O(n^2)$. The outer loop iterates $(\text{length} - 1)$ times, and the inner loop iterates up to j times for each outer iteration. Although it breaks out of the loop early if no swaps are made. But the worst-case scenario occurs when the array is initially in reverse order, requiring a maximum number of iterations. While these optimized versions reduce unnecessary iterations compared to the standard Bubble Sort, their worst-case time complexity remains quadratic.

2.No.

Both versions of the Bubble Sort optimized algorithms you provided have the same worst-case time complexity, which is $O(n^2)$.

3.Yes.

We can Look for Nested Loops to quickly analyze the worst-case time complexity. Each level of nesting usually multiplies the time complexity. Take the most time within the loop that determines the complexity.