5. Finding Complexities

Symbol	Name	Explain
O(1)	Constant Time	The Algorithm runtime does not changes with input size.
O log(n)	Logarithmic Time	The Algorithm reduces the problem size each time, such as Binary Search.
O(n)	Linear Time	The Algorithm's runtime increases proportionally to the input size.
O(n log n)	Linearithmic Time	Commonly seen in efficient sorting algorithms like Merge Sort.
$O(n^2)$	Quadratic Time	Typically seen in algorithms that involve nested loops, like Bubble Sort.
$O(2^n)$	Exponential Time Complexity	The run time doubles with each addition to input.
O(n!)	Factorial Time Complexity	Seen in algorithms that involve generating all permutation, such as the Travelling Sales Man Problem.

How to Find Time Complexities?

Step 1 : Identify the loops and recursive calls in your algorithm.

- If the algorithm has a single loop that runs n times, then complexity will be O(n).
- If there is nested loop, then time complexity will be $O(n^2)$, $O(n^m)$ or higher.

Step 2 : For each operation, figure out how many times it runs as the input grows.

- Constant time operations (like simple arithmetic) are O(1).
- Recursive algorithms may have more complex time complexities that depend on the depth of recursion.

Step 4 : Drop Constants and non-dominant terms.

- If you have O(n+100), its simply O(n) because constants doesn't matter in Big O notation.
- For Example, $O(n^2 + n)$ is simplified to $O(n^2)$ because the quadratic term dominates.