**Big O Notations**

**1. O(1) - Constant Time**

* **Description:** The runtime is constant and does not depend on the input size.
* **Example:** Accessing an element in an array by index.
* **Performance:** Very fast.

**2. O(log n) - Logarithmic Time**

* **Description:** The runtime grows logarithmically as the input size increases. Typically occurs when the problem size is reduced by a constant factor (e.g., halving) each step.
* **Example:** Binary search.
* **Performance:** Efficient, even for large inputs.

**3. O(n) - Linear Time**

* **Description:** The runtime grows directly in proportion to the input size.
* **Example:** Iterating through an array or list.
* **Performance:** Reasonable for most tasks.

**4. O(n log n) - Log-Linear Time**

* **Description:** Combines linear and logarithmic growth. Common in "divide and conquer" algorithms like sorting.
* **Example:** Merge sort, quicksort (average case).
* **Performance:** Efficient for large inputs.

**5. O(n²) - Quadratic Time**

* **Description:** The runtime grows proportionally to the square of the input size. Often seen in algorithms with nested loops.
* **Example:** Bubble sort, comparing all pairs in a list.
* **Performance:** Gets slow quickly as input grows.

**6. O(n³) - Cubic Time**

* **Description:** The runtime grows proportionally to the cube of the input size. Seen in algorithms with three nested loops.
* **Example:** Checking all triplets in a dataset.
* **Performance:** Very slow for anything but small inputs.

**7. O(2ⁿ) - Exponential Time**

* **Description:** The runtime doubles with each additional input. Often seen in brute-force solutions to problems like the traveling salesman.
* **Example:** Solving the traveling salesman problem by trying all permutations.
* **Performance:** Impractical for large inputs.

**8. O(n!) - Factorial Time**

* **Description:** The runtime grows as the factorial of the input size. Often seen in algorithms that involve generating all permutations.
* **Example:** Solving a puzzle by exploring all possible configurations.
* **Performance:** Infeasible for anything but the smallest inputs.

**Common Graph for Visualizing Growth Rates**

From fastest to slowest:

1. **O(1)** → Flat line.
2. **O(log n)** → Grows slowly, like a curve flattening out.
3. **O(n)** → Straight diagonal line.
4. **O(n log n)** → Slightly steeper than O(n).
5. **O(n²)** → Steep parabola.
6. **O(2ⁿ)** and **O(n!)** → Skyrocket rapidly.

**Summary**

* **Efficient Complexities:** O(1), O(log n), O(n), O(n log n).
* **Less Efficient Complexities:** O(n²), O(2ⁿ), O(n!).

Choose algorithms with lower complexities whenever possible, especially for larger inputs!