TIME COMPLEXITY

What is Time Complexity?

Time complexity means **how long your code takes to run** when the input becomes bigger. It helps you understand if your code is **fast or slow**

Big-O Notation

We use **Big-O** to show time complexity. It shows how your code behaves when the input grows.

Common Types of Time Complexity:

1. O(1) — Constant Time

Always takes the same time, no matter the input.

```
def show_first_item(items):
   print(items[0])
```

2. O(n) — Linear Time

Time increases with input size.

```
def print_all_items(items):
   for item in items:
       print(item)
```

3. O(n^2) — Quadratic Time

A loop inside a loop. Gets slow fast!

```
def print_all_pairs(items):
   for a in items:
       for b in items:
           print(a, b)
```

4. O(log n) — Logarithmic Time

Cuts the problem in half each time. Very fast!

```
def binary_search(items, target):
    low = 0
    high = len(items) - 1

    while low <= high:
        mid = (low + high) // 2
        if items[mid] == target:
            return mid
        elif items[mid] < target:
            low = mid + 1
        else:
            high = mid - 1

    return -1
```

5. O(n log n) — Linearithmic Time

A mix of linear and logarithmic. Found in many sorting algorithms.

```
# Example: Using Python's built-in sorted() which is O(n log n)
numbers = [4, 2, 7, 1]
sorted_numbers = sorted(numbers)
print(sorted_numbers)
```

How to Know the Time Complexity?

Code Feature	Time Complexity
Simple step	O(1)
One loop	O(n)
Nested loops	O(n^2)
Dividing input	O(log n)
Sorting	O(n log n)

Some Useful Tips:

- Use **sets** and **dictionaries** instead of lists when you need to **search fast**.
- Avoid nested loops (O(n^2)) for large data.
- Use built-in functions like sorted(), max(), min() — they are optimized.

Time vs Space Complexity:

- **Time Complexity** → How fast your code runs.
- **Space Complexity** → How much memory your code uses.

Example:

```
def double_list(lst):
   new_lst = []
   for item in lst:
      new_lst.append(item * 2)
   return new_lst
```

- Time: O(n)
- Space: O(n) (because it creates a new list

Final Notes:

- Try to write efficient code for big input.
- Learn to **analyze loops** and **recursive functions**.
- Use **Big-O notation** to compare algorithms.

Presented By:

Narmeen Asghar