Validate IP Address - Full Documentation

1. **Problem Statement**

   Given a string queryIP, return:

   - "IPv4" if queryIP is a valid IPv4 address,
   - "IPv6" if it is a valid IPv6 address,
   - "Neither" if it is not a valid IP address of either type.

   IPv4 Format:

   - Consists of 4 decimal numbers separated by dots (.)
   - Each part must be in the range 0–255 (inclusive)
   - No leading zeros allowed (e.g., "192.168.01.1" is invalid)

   IPv6 Format:

   - Consists of 8 groups of 1 to 4 hexadecimal digits, separated by colons (:)
   - Hex digits include 0-9, a-f, A-F
   - Leading zeros are allowed
   - Groups must not be empty and must contain only valid hex characters

2. **Intuition**

   The task can be simplified by first detecting the format using the presence of . or :

   - If . is found, try to validate as IPv4.
   - If : is found, try to validate as IPv6.
   - If neither is a match, return "Neither".

3. **Key Observations**

- A valid IPv4 should have exactly 4 parts when split by .
- A valid IPv6 should have exactly 8 parts when split by :
- Leading zeros invalidate an IPv4 segment but not an IPv6 one
- IPv6 segments must be between 1 and 4 characters, all valid hexadecimal

4. **Approach**

- Check for IPv4:
  - Split by ., ensure 4 parts
  - Validate each part is:
    - Numeric
    - In range [0, 255]
    - No leading zero unless it's "0"
- Check for IPv6:
  - Split by :, ensure 8 parts
  - Validate each part:
    - Length is between 1 and 4
    - All characters are hexadecimal
- If neither format passes validation, return "Neither"

5. **Edge Cases**

- "192.168.1.00" → Invalid (leading zero)
- "192.168.1.1." → Invalid (trailing dot)
- "256.256.256.256" → Invalid (exceeds 255)
- "2001:0db8:85a3::8A2E:037j:7334" → Invalid (contains j)
- "02001:0db8:85a3:0000:0000:8a2e:0370:7334" → Invalid (group too long)

6. Complexity Analysis

Time Complexity

- IPv4 and IPv6 validation both operate in O(1) time, as the number of segments is fixed (4 or 8), and maximum character count is small.

Space Complexity

- O(1) — only a few strings and lists of size at most 8 are created during splitting.

7. Alternative Approaches

a. Regular Expressions

- Use regex patterns to match IPv4 and IPv6 formats.
- Pros: Concise
- Cons: Less readable, harder to debug, may not catch subtle edge cases

b. Python's ipaddress module

- ipaddress.ip_address(queryIP)
- Pros: Very simple, reliable
- Cons: Doesn't help in understanding internal rules

8. Test Cases

Input	Output	Explanation
"172.16.254.1"	IPv4	Valid IPv4
"256.256.256.256"	Neither	Values exceed 255
"192.168.01.1"	Neither	Leading zero invalidates the IPv4 part
"2001:0db8:85a3:0:0:8A2E:0370:7334"	IPv6	Valid IPv6
"2001:0db8:85a3::8A2E:037j:7334"	Neither	Invalid character 'j' in IPv6

"02001:0db8:85a3:0000:0000:8a2e:0370:7334"	Neither	Group too long in IPv6

9. Final Thoughts

- This problem is a great test of string manipulation and pattern recognition.
- Manually implementing checks builds a strong understanding of IP formatting.
- While libraries and regex can simplify the task, coding logic manually ensures better control over validation and error handling.