Q 1. Difference between sort and sorted? Find the if 2 string is anagram.

Ans - Both sort and sorted are used to sort lists, but they have key differences in terms of usage and behavior. Below are those differences.

* Sort –
  + **In-place** Sorting: The sort of method sorts the list in place, meaning it modifies the original list and does not return a new list.
  + Usage: It is a method of the list object, so it can **only be used on lists**.
  + Return Value: It returns **None**.
  + Syntax: list.sort(key=None, reverse=False
* Sorted –
  + **Creates a New List**: The sorted function returns a new sorted list from the elements of any iterable (not just lists), leaving the original iterable unchanged.
  + Usage: It can be used on any iterable, including **lists, tuples, strings, and dictionaries**.
  + Return Value: It returns **a new list**.
  + Syntax: sorted(iterable, key=None, reverse=False)

To determine if two strings are anagrams in an efficient way, you can use the following approach:

1. Sort and Compare: This method involves sorting both strings and then comparing them. This has a time complexity of 𝑂(𝑛log⁡𝑛)O(nlogn).

2. Count and Compare: This method involves counting the occurrences of each character in both strings and then comparing these counts. This can be done in 𝑂(𝑛)O(n) time complexity if using a fixed-size character set like ASCII.

Method 1: Sort and Compare

def are\_anagrams\_sort\_and\_compare(s1, s2):

# If lengths of both strings are not equal, they cannot be anagrams

if len(s1) != len(s2):

return False

# Sort both strings and compare

return sorted(s1) == sorted(s2)

# Example usage

s1 = "listen"

s2 = "silent"

print(are\_anagrams\_sort\_and\_compare(s1, s2))

# Output: True

Method 2: Count and Compare

from collections import Counter

def are\_anagrams\_count\_and\_compare(s1, s2):

# If lengths of both strings are not equal, they cannot be anagrams

if len(s1) != len(s2):

return False

# Count characters in both strings and compare

return Counter(s1) == Counter(s2)

# Example usage

s1 = "listen"

s2 = "silent"

print(are\_anagrams\_count\_and\_compare(s1, s2)) # Output: True

Explanation

1. Sort and Compare:

• Sort both strings and check if they are equal.

• This is straightforward and easy to understand but involves sorting which is 𝑂(𝑛log⁡𝑛)O(nlogn).

1. Count and Compare:

• Use a Counter from the collections module to count the occurrences of each character in both strings.

• This method is more efficient with a time complexity of 𝑂(𝑛)O(n), making it preferable for longer strings.

Both methods will work efficiently for typical cases, but for very large strings or performance-critical applications, the counting method (Method 2) is generally more efficient.