## Week-01 Algo Questions

**Question-1 Valid Anagram**

Given two strings ***s*** and ***t***, return true if ***t*** is an anagram of ***s***, and false otherwise.

An *Anagram* is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

*Constraints:*

1 <= s.length, t.length <= 5 * 104

s and t consist of lowercase English letters.

***Follow up:*** What if the inputs contain Unicode characters? How would you adapt your solution to such a case?

*Example 1:*

Input: s = "anagram", t = "nagaram"

Output: true

*Example 2:*

Input: s = "rat", t = "car"

Output: false

*Please, test your solution here :* https://leetcode.com/problems/valid-anagram/

**Question-2 Valid Palindrome**

Write a function that accepts a string and determines whether it is a palindrome, ignoring non-alphanumeric characters and case and returns true if it is palindrome.

(A string is said to be palindrome if it reads the same backward as forward, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters).

*Examples:*

*Input: Do geese see God?* *Output: True*

*Input: Was it a car or a cat I saw?* *Output: True*

*Input: A brown fox jumping over* *Output: False*

***Constraints:***

- *1 <= s.length <= 2 * $10^5$*
- *s consists only of printable ASCII characters.*

*Please, test your solution here :* https://leetcode.com/problems/valid-palindrome/

*You may try two pointer technique for second question to improve your time & space complexity.*

**Two Pointer Strategy / Technique**

- The name, “two pointers”, explains the strategy clearly.
- It's the use of two different pointers (usually to keep track of array, ArrayList or String indices) to solve a problem with the benefit of saving time and space.
- A pointer is a reference to an object.
- In many problems involving collections such as arrays or lists, we should analyze each element of the collection compared to its other elements.
- We can process two elements per loop instead of just one. Common patterns in the two-pointer approach entail:
  - Two pointers, each starting from the beginning and the end until they both meet.
  - One pointer moving at a slow pace, while the other pointer moves at twice the speed.