STRINGS

String is an array of characters.

vector <char> str. ×   '\n'   '\t'

String:

→ Array has a pre-defined size and through that size, we get to know what the last element in an array is.

→ Strings are terminated by a special character and that special character defines the end of a string.
(special character: '\0')
[NOTE: Valid for C / C++]

→ In strings, maximum of 256 unique characters can be present.
↓
ASCII characters

O(1)

⇒ (a, b, c, ..., z) (A, B, C, ..., Z) (@, !, #, $ ...)

→ Check if lowercase (ch):
(int) 'a'
if ((int) ch >= 97 && (int) ch <= 122) ✓
if (ch >= 'a' && ch <= 'z') ✓

→ Check if uppercase (ch):
if (ch >= 'A' && ch <= 'Z')

→ Check if digit (ch):
if (ch >= '0' && ch <= '9')

→ Convert lowercase (ch) to its corresponding uppercase:
'a' → 'A'   'n' → 'N'

= (ch − 32)
= (ch − ('a' − 'A')) = $\boxed{ch - 'a' + 'A'}$ ✓

→ Uppercase to lowercase:
⇒ (ch − 'A' + 'a')

[ch + ('a' − 'A')
ch − ('A' − 'a')]

Palindrome: A string equivalent to its reverse is called a palindromic string.

"abc" ×
"aba" ✓

"abcdcba" ⇒ True.

"abcdba" ⇒ False

`"A man, a plan, a canal: Panama"` ✓

"amanaplanacanalpanama"

TC: O(n)
Aux.space: O(n)

Anagrams: "act", "tac" → ✓
"aac", "cca" → ×

#1 Sort the strings:
If two strings are anagrams, then their sorted versions will be the same.

"act", "tac"
↓ ↓
(act == act) → True.

TC: O(n log n)
Aux.space: O(1)

#2: 2 Frequency maps:
maps will take O(26) = O(1) amount of space.

act	tac
a → 1	a → 1
c → 1	c → 1
t → 1	t → 1

True

aac	cca
a → 2	a → 1
c → 1	c → 2

False.

[~~ab~~ → ~~abc~~]

TC: O(n + n + 26) = O(n)
Aux. space: O(52) = O(1)

#3: 1 Frequency map

act
tac

a → ~~1~~ 0
c → ~~1~~ 0
t → ~~1~~ 0
True.

aac
cca

a → ~~2~~ 1
c → ~~1~~ −1
False.

TC: O(n + n + 26) = O(n)
Aux. space: O(26) = O(1)

a[i] = 'a' ⇒ (a[i] − 'a') = 0
a[i] = 'b' ⇒ (a[i] − 'a') = 1
⋮
a[i] = 'z' ⇒ (a[i] − 'a') = 25

Reverse a strings: "geeks"
↳ "skeeg"

i . like, this . program

TC: O(n)
Aux.space: O(n)

word = i ~~like~~ ~~this~~ . ~~program~~

[i, like, this, program]