$\Rightarrow [1, 3, 5, 9, 14]$ target = 12

$\frac{4 + 23}{2} = \frac{27}{2} = \boxed{13}$

STRINGS

like a character array

['g', 'e', 'e', 'k', 's'] size()

strings "geeks" length()

Strings are terminated with a special character ('\0')

'\n' → endline

'\t' → tab-characters

'\0' → string terminator

"geeks" → internally stored as "geeks\0"

"geeks" → "ge"

Int 'a'

We have a distinct ASCII value associated with each character.

char ch = 'a'

print ((int) ch) → Print the ASCII value of ch = 97.

# check if a given character is lowercase English alphabet

if (ch >= 97 && ch <= 122)

'a' → 97
⋮
'z' → 122

if (ch >= 'a' && ch <= 'z')

# Check if Uppercase:

if (ch >= 'A' && ch <= 'Z')

# check if digit

if (ch >= '0' && ch <= '9')

# Convert a given lowercase character to corresponding uppercase.

if ch = 'b' ⟹ ch = 'B'

ch = 'y' ⟹ ch = 'Y'

Soln: ch = ch - 32

ch = ch - 'a' + 'A'

= ch - 97 + 65 = ch - 32

$\boxed{ch = ch - 'a' + 'A'}$

ch	ch - 'a'	ch - 'a' + 'A'
a	0	0 + 'A' = 'A'
b	1	1 + 'A' = 'B'
c	2	2 + 'A' = 'C'
d	3	3 + 'A' = 'D'
⋮	⋮	⋮
z	25	25 + 'A' = 'Z'

# Convert uppercase to corresponding lowercase:

$\boxed{ch = ch - 'A' + 'a'}$

Palindrome: Something which is the same when read forward or backward.

"abba" → Palindrome

"abca" → Not Palindrome

"abab"

1 2 3 2 → X

1 2 1 → ✓

Approach-1

1) Create a copy & reverse it.
2) Compare the original & reversed version.

TC = O(n)
Aux space = O(n)

Approach-2

Two-pointer approach

"abba" → True

"abca" → False

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

"A man, a plan, a canal: Panama" ⟹ True.

"race a car" 'e' ≠ 'a' ⟹ false.

"aba"

Ch >= '0' && ch <= '9' || ch >= 'a' && ch <='z' || ch >='A' && ch <='Z'

ANAGRAMS

s1 = "abcd" s2 = 'abcdd' X

s1 = "abbc" s2 : "abcb" ✓

TC

Sorting approach → O(n log n)

Hashmap - approach → O(n)

↳ Storing freq. of all chars in the strings: O(n)

Comparing freq. of each character: O(26) = O(1)

s1 = abbc
s2 = abcb

s1	s2
a → 1	a → 1
b → 2	b → 2
c → 1	c → 1
d → 0	d → 0
e → 6	e → 0
⋮	⋮
z → 0	z → 0

	0	1	2	3		25
freq_s1	5	7	3	0	---	7
freq_s2	5	7	3	0	---	7
char_freq	+5 -5	+7 -7	+3 -3	+0 -0	---	+7 -7

If all values in 'char_freq' are equal to zero in the end. Then both strings are anagrams.

"geeks" → "skeeg"

↓

seekg

↓

"skeeg"

i. like. this. program. very. much

hcum. yrev. margorp. siht. ekil. i

much. very. program. this. like. i

["i", "like", "this", "program", "very", "much"]

"i" ~~like~~ ~~this~~ ("much")

"much. very. program. this. like. i"

LEXICOGRAPHICAL RANK

(acb)

abc → 1
acb → 2
bac → 3
bca → 4
cab → 5
cba → 6

n * n!

a b c d e f g h i j k l m n o p q r s t u v w x y z

① view

→i - - -
→e - - -] 2 × 3! = 12

② view

→e - -] 1 × 2! = 2

① view

0 × 1! = 0

① view

0

Overall, 12 + 2 + 0 = 14 strings are lexicographically smaller than "view".

⟹ Lexicographical rank of "view" = 14 + 1 = 15.

For a string of length 'n':

Rank = (number of chars smaller than the first char) * (n-1)!
+ Rank (s[1 … n-1])

"o"

s = view
s_sorted = eivw

s = veiw
s_sorted = eivw

veiw [e - - -
i - - -] 2 × 3! = 12

eiw 0 × 2! = 0

iw 0 × 1! = 0

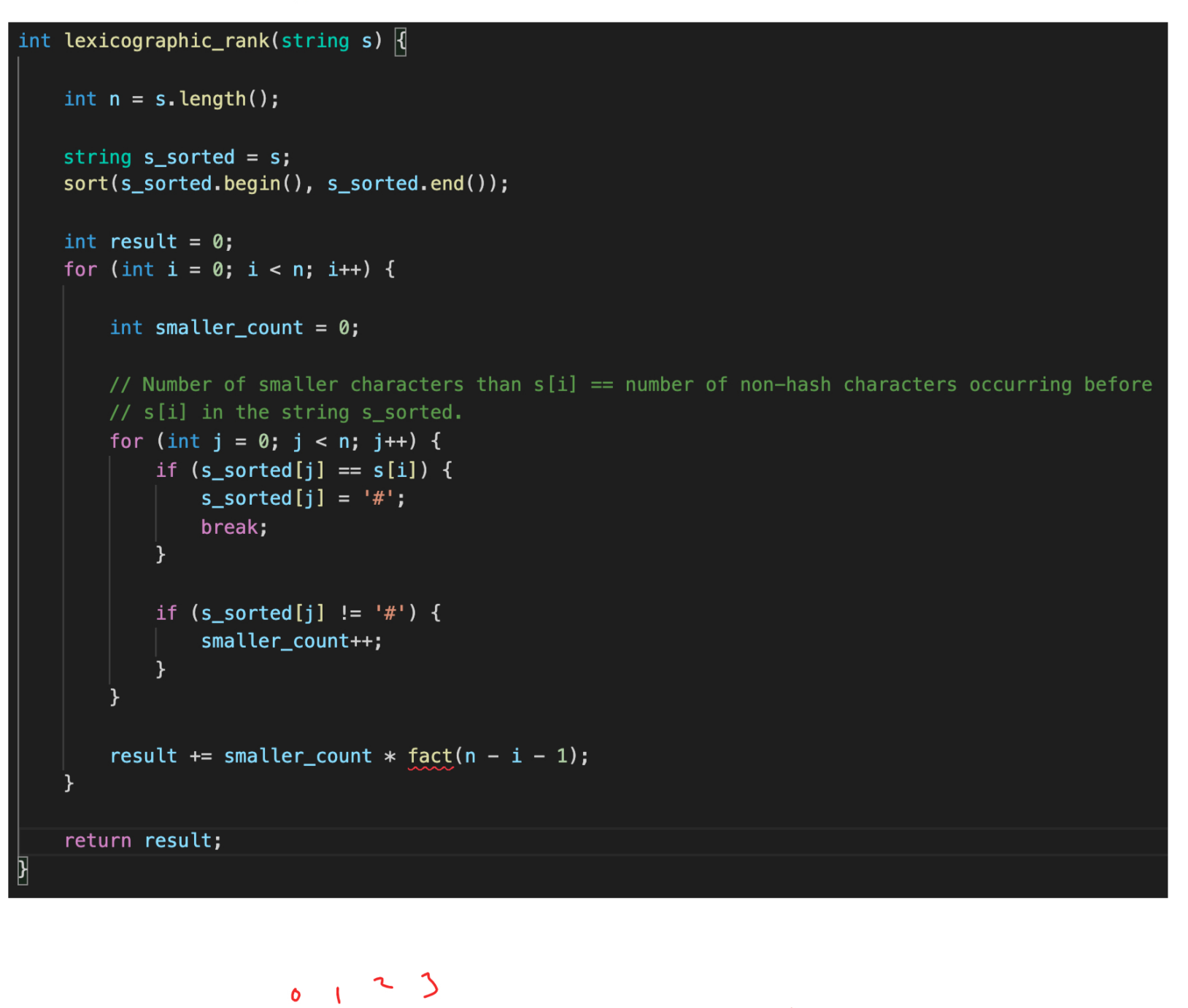
w 0 × 0! = 0

12 + 0 + 0 + 0 ~ 12

⟹ RANK = 12 + 1 = 13.

0 1 2 3
veiw (n-i-1)

[smaller_count × (n-i-1)!]



```
int lexicographic_rank(string s) {

    int n = s.length();

    string s_sorted = s;
    sort(s_sorted.begin(), s_sorted.end());

    int result = 0;
    for (int i = 0; i < n; i++) {

        int smaller_count = 0;

        // Number of smaller characters than s[i] == number of non-hash characters occurring before
        // s[i] in the string s_sorted.
        for (int j = 0; j < n; j++) {
            if (s_sorted[j] == s[i]) {
                s_sorted[j] = '#';
                break;
            }

            if (s_sorted[j] != '#') {
                smaller_count++;
            }
        }

        result += smaller_count * fact(n - i - 1);
    }

    return result;
}
```

s = veiw n = 4

s_sorted = ##~~w~~#

i = ~~0~~ ~~1~~ ~~2~~ 3

smaller_count = ~~0~~ ~~1~~ ~~2~~ 0

j = ~~0~~ ~~1~~ ~~2~~ 0

result = 0 +
2 * 3! +
0 * 2! +
0 * 1! +
0 * 0!
= 12

RANK = 12 + 1

n

(n-1)!

(n-2)!

5 % 2 = 1

8 % ③ = 2

x % m ∈ [0, m-1]