class 8 Strings

8) Jewels & Stones

Jewels = aA (N)

Stones = aAAAbbB (M)

4 jewels

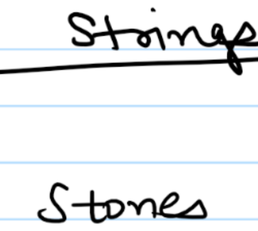
Brute force:

```
for (char ch : stones)
    for (char cha : jewels)
        if (ch == cha)
```

TC: O(M*N)

SC: O(1)

2nd approach:

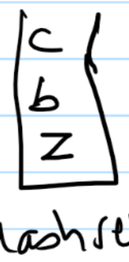


→ Hashset characters in jewels

Stones	a	A	A	A	b	b	B
count	1	2	3	4	4	4	4 → 4 Ans

M

jewels: z b c



Hashset

N

stones: a b z Z

count: 0 1 2 2 → Ans = 2

TC: O(M+N)

SC: O(M)

82 Valid anagram

"abcd" → "bdac" ✓

"rat" → "dog" ✗

Approach 1 Sort both the strings & check if they are equal or not

abcd —sort→ abcd

bdac —sort→ abcd

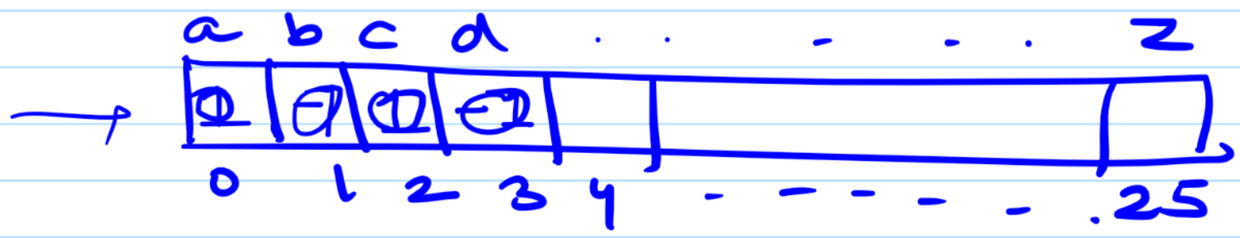
] = ? True

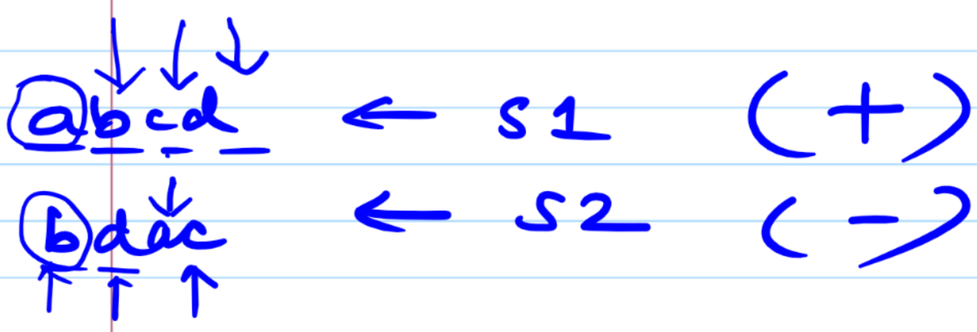
TC N log n

Approach 2

abcd bcda

False





Algo: 1) we can count occurence of each letter in 2 strings. we'll maintain an array of size 26

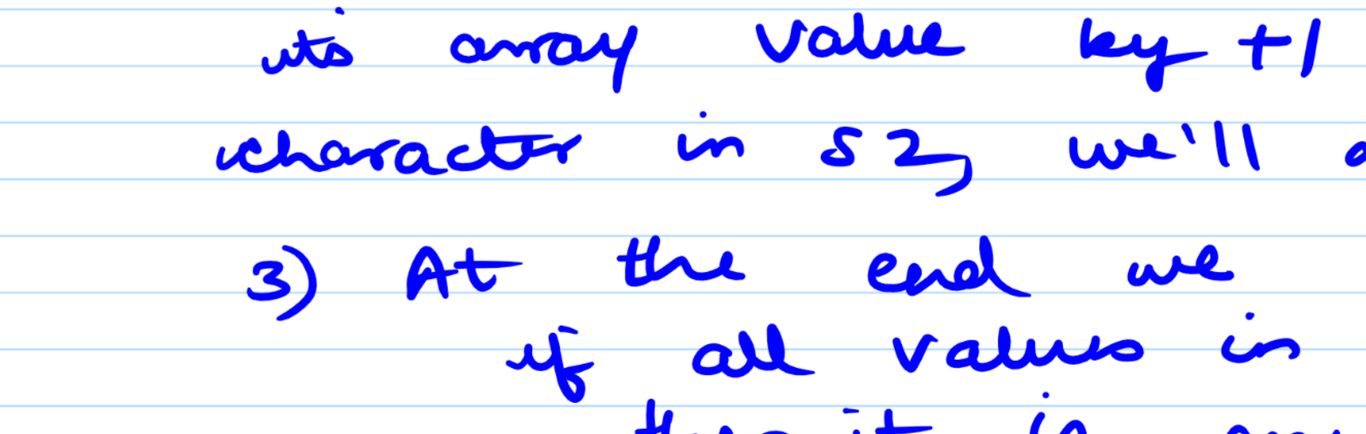
2) For each character in s1, we'll increment its array value by +1 & for each character in s2, we'll decrement its value

3) At the end we check if all values in array == 0, then it is an anagram

TC: O(N)

SC: O(1)

3rd app: HashMap



Q3: Valid palindrome

A a

A = a True.

Approach 1: Reverse the string

TC: O(N)

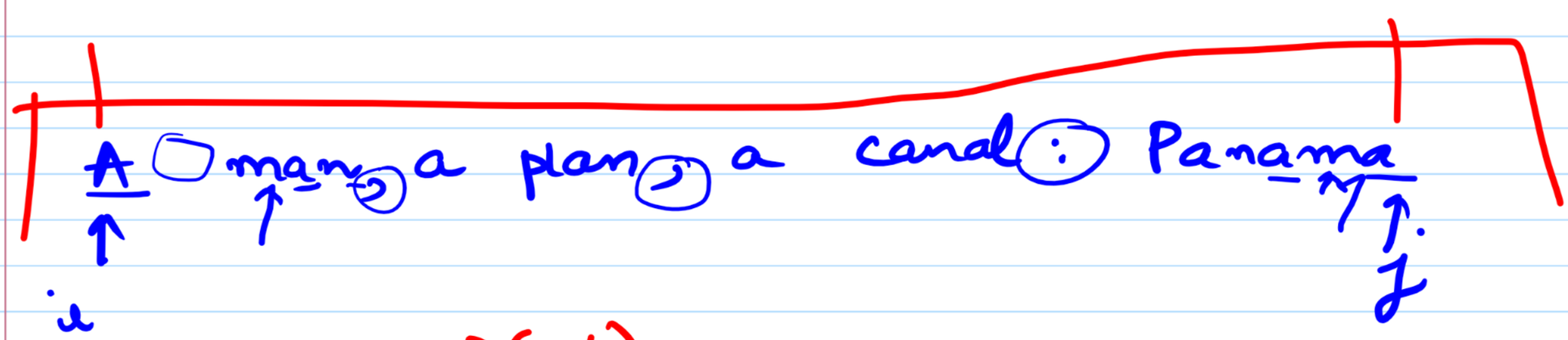
Approach 2: 2 pointer approach

Algo: 1) set 2 pointers, one at each end of the string.

2) If the input is a palindrome, then both the pointers should point to equivalent characters if this condition is not met, return false.

3) We'll ignore special characters while traversing.

4) Continue until both the pointers meet



TC: O(N)

SC: O(1)

aA = aa = A → Palindrome ✓

94: Redistribute characters to make strings equal

1) "abc", "aabc", "bc" = 3.

"abc", "abc", "abc" → True.

2) ab b bb → false

a = 1
b = 3

1 % 3 ≠ 0

3) cabcd, dabcab, d

abcd , dabcab , cd

abcd , dabc , abcd → True

abcd , abcd ,

aab, ab, aaab → True

~~a →~~

aab, aab, aab

a: 2 + 2 + 2 = 6

b: 1 + 1 + 1 = 3

Frequency / No. of words

→ ab aab aab

5 % 3 != 0

a: 1 + 2 + 2 = 5

b = 1 + 1 + 1 = 3

6 / 3 ⑥ % 3 == 0

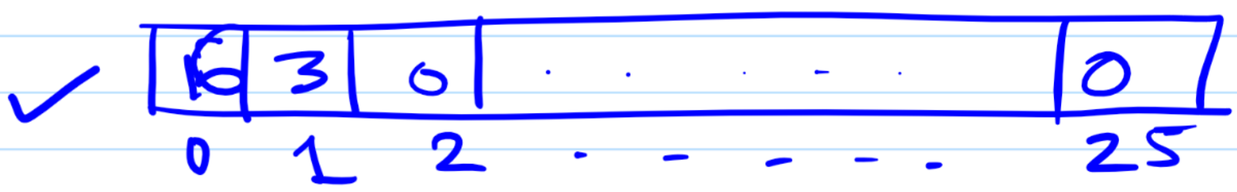
— — —

Total Frequency of each letter % No. of words = 0

TC: O(N) N is sum of length of all strings

SC: O(1)

aab ab aaab



6 % 3 == 0 ✓ true

3 % 3 == 0 ✓

Algo:

1) Create an array of 26 size which will store the frequency of each character.

2) Iterate over all the words & calculate the total frequency of each character

3) If frequency % total no. of ≠ 0 for any

words

character, return false.

4) Return true.

ab , a , b

3

2 % 3 == 0 ✗ Not possible

Q5: Split a string in balanced strings

"RLRRLLRLRL"

① No. of R == No. of L = 4

RL , RRLL , RLRL , LRLR

② "RLRRRLLRLL"

RL, RRRLLRLL

③ LLLLRRRL

RLRL res=0 count=0

L → +1

R → -1

count	-1	0	-1	0
	R	L	R	L
res		1		2

2 possible balanced substrings

RL, RLRL

LR ✗

R L | R R L L | R L | R L

① ② ③ ④

	R	L	R	R	L	L	R	L	R	L
count	-1	0	-1	-2	-1	0	-1	0	-1	0
res		1				2		3		4

TC: O(n)

SC: O(1)

Q6: Reverse vowels

Approach: 2 pointer.

Algorithm: 1) Maintain 2 pointers left = 0, right = end of the string (n-1)

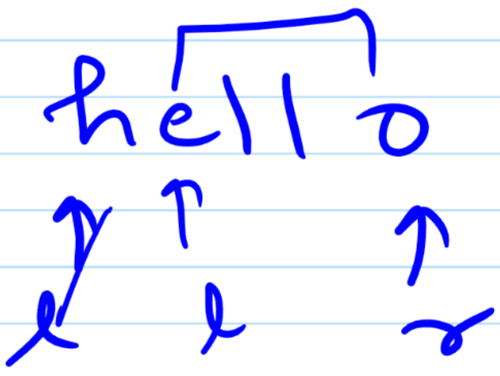
2) keep incrementing left pointer till it points to a vowel.

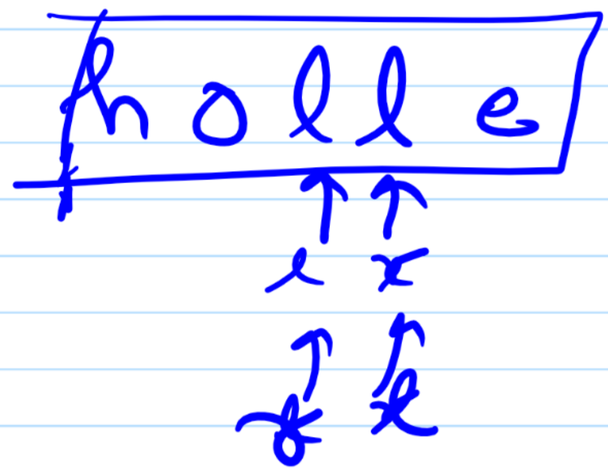
3) keep decrementing right pointer till it points to a vowel

4) swap values at left & right

5) increment left, decrement right

6) Return string s





TC: O(m).

SC: O(1)