Remove Duplicate

Wed 7/20

Input: Sorted array ; Output : int.

A naive approach is to use a hashmap to check uniqueness $\Rightarrow$ $O(n)$ runtime, space $O(\#\text{ of unique})$

We can use two pointers and swap positions of the elem. if the subseq. elem. is same. and keep count.

2 , 3 , 3 , 3 , 6 , 9 , 9 count = 1

count = 2

count = 3

count = $\boxed{4}$

```
def f(A):
    left, right, count = 0, 0, 1   → first elem is uniq.
    while right < len(A):
        if A[left] ≠ A[right]:
            count += 1
            left = right
            right += 1
        else:
            right += 1
    return count
```

8 min total

Input: Unsorted array and integer.
Output: integer

Remove key $\in \mathbb{Z}$ from the array.

The Naive Approach is to traverse the array and check each elem. and use a counter.

↳ We can expand on this where we swap position for each elem that is equal to the key input value.

3 2 3 6 3 10 9 3

↑ l ↑ r

A[l] = key

swap
A[l] = A[r]
A[r] = A[l]
r - -

3 2 3 6 3 10 9 3

↑ l ↑ r

9 2 3 6 3 10 3 3

↑ ↑ ↑

loop until l < r

9 2 10 6 3 3 3 3

↑ ↑ ↑ ↑

And keep count

6min

```
def f(A, k):
    l, r = 0, len(A) - 1
    count = 0
    while l < r:
        if A[l] == k:
            ~~count += 1~~
            A[l] = A[r]
            A[r] = A[l]
            r -= 1
        else:
            ~~l += 1~~
            count += 1
            l += 1
    return count
```

Tmin

4/20

Squaring Sorted Arr.

Input: Arr
Output: Arr

The Naive approach would be to take each elem., square it, and then use a sorting algorithm.
↳ $O(n \log n)$ runtime, $O(1)$ space.

$\boxed{\text{Ex1}}$ $-2, -1, 0, 2, 3$

l c r

It's a sorted array, so we can have ptr's on both sides and compare those elem.
↳ If the left ptr elem is greater

$-2 \quad -1 \quad 0 \quad 2 \quad 3$

l r

- r ptr decrement for each step.
- We swap elem when $A_l^2 > A_r^2$
- While $l < r$

20min

$O(n), O(1)$

$l = 0, r = len(A) - 1$

~~Left~~ While $l < r$

IF $A_l^2 > A_r^2$

$A_r = A_l^2$

$A_l = A_r$

Had to use Python's Tuple Assign.

$r--$

IF $A_l^2 \leq A_r^2$

$A_r = A_r^2$

$r--$

return A

22 min Design, 33 min code pass

- How can I recognize to start the list from the back?

(*) Spent too much time looking for $O(1)$ space sol'n.

1 hr 58 min

Triplet Sum to Zero 7/20

Input: List[int], Out: List[List[int]]

Bruteforce: $O(n^3)$, $O(n)$

There is probably an $O(n^2)$, $O(n)$ sol'n.

Need to traverse whole arr. → $O(n)$ runtime.

[-3, 0, 1, 2, -1, 1 -2]

↑ ↑

We can take a running sum.

→ We can take each value in the list and check to find the value that would add to 0 (call it x) ($-3 + x = 0 \iff x = 3$)

Then we can use ptr's, from left and right, to find two values that would sum to x.

↳ Initial pointers would be on $i+1$ and $len(l)-1$.
(If using a for i loop)

$-3 + x = 0 \iff \underline{x = 3}$

$\boxed{\text{Ex}}$ $-3 \quad 0 \quad 1 \quad 2 \quad -1 \quad 1 \quad -2$

$A_i \quad A_l \quad A_r$

(*) It would be easier if sorted
↳ $O(n^2 \log n)$ time

This was the approach to take

~~For i in range(|A|)~~
~~X = abs(A_i)~~

We can take two elem at a time while traversing the list and find the value that would add to 0.
↳ Use Python in ()?

0	1	2	3	4	5	6
-3	0	1	2	-1	1	-2

We can swap positions for elem. in index 0 and 1

0	1	-3	1	2	-1	1	2

22 min Quit

Triplet Sum close to target 7/20

Given unsorted array, and a target number, return the sum of triplet close the target as possible.

I/O : Arr / int

- Prob need to use abs(sum-target)
- Need to sort → $O(n \log n)$ where $n = |A|$

A Naive (BF) Approach is to check every combination of elem and take the difference of the triplet sum and target. Then return the ~~smalles~~ sum w/ smallest diff. → $O(n^3)$, $O(1)$

A better approach will probably take $O(n^2)$, $O(1)$

target = 2

$\boxed{\text{Ex1}}$ $[-2, 0, 1, 2] \xrightarrow{\text{sort}} [-2, 0, 1, 2]$

$\uparrow A_i \quad \uparrow A_l \quad \uparrow A_r$

Similar to prev. prob. we take an elem and take the sum w/ the other two elem and keep track of the minimum total.

$\rightarrow A_i + A_l + A_r = -2 + 1 + 2 = 1 = \text{curr\_sum}$

abs(target − curr_sum) = 1 = min_diff

$\boxed{\text{Ex2}}$ $[-3, -1, 1, 2]$, tar = 1

$\uparrow A_i \quad \uparrow A_l \quad \uparrow A_r$

c_s = −2
min_diff = 3

$\rightarrow$ ~~[-3, -1, 1, 2]~~ ~~c_s = 2~~ ~~min_diff = 1~~

$\uparrow A_i \quad \uparrow A_l \quad \uparrow A_r$

(*) Need to use while loop inside the for

[-3, -1, 1, 2]

$\uparrow \quad \uparrow \quad \uparrow$

c_s = 0
min_diff = 1

$\uparrow \quad \uparrow \quad \uparrow$

c_s = 2
min_diff = $\boxed{1}$

16min

```
import math

def func(A, t):
    A.sort()
    min_d = math.inf
    for i in range(len(A)):
        l, r = i+1, len(A) - 1

        while l < r:
            c_s = A[i] + A[l] + A[r]
            c_d = abs(t - c_s)
            if min_d > abs(t - c_s)
                min_d = c_d
                l, r = l+1, r-1
            while l<r and A_l == A_l-1 :
                l+=1
            while l<r and A_r == A_r+1 :
                r-=1
            else :
                l+=1

    return c_s


Umin qgnit
```

Triplet Sum to Zero

7/20

- Need to use sorting. $\Rightarrow O(n^2), O(n)$
- Use helper f'n.
- Use a running sum $\Rightarrow$ current_sum $= A[l] + A[r]$

[-3, 0, 1, 2, -1, 1, -2]

$\Rightarrow$ [-3, -2, -1, 0, 1, 1, 2]

A_i A_l A_r

curr_sum = -3
target_sum = 3

l r

c_s = -2
t_s = 3

l r

$\Rightarrow$ [-3, 1, 2]

End of helper f'n

A_i l r

5|min