

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 = 4

def f(A):

left, right, count = 0, 0, 1 \rightarrow first elem is uniq.

while right < len(A):

if A[left] \neq A[right]:

count += 1

left = right

right += 1

else:

right += 1

return count

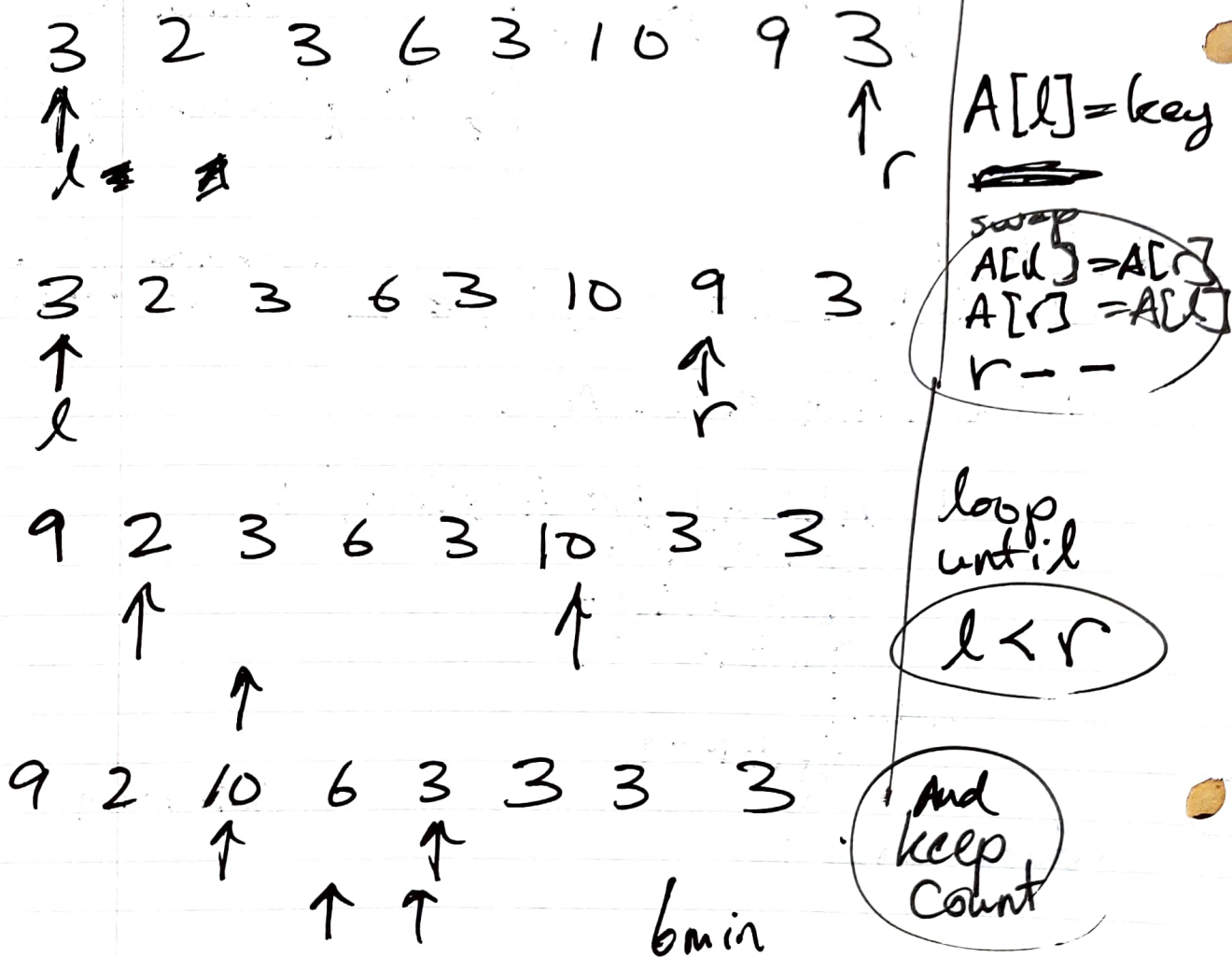
8 min total

Input: Unsorted array and integer.
Output: integer

Remove $\text{key} \notin \mathbb{Z}$ 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.



```
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:
```

```
            count += 1
```

```
            count += 1
```

```
            l += 1
```

```
    return count
```

T_{min}

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.
 $\rightarrow O(n \log n)$ runtime, $O(1)$ space.

Ex 1 -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.
 \rightarrow If the left ~~ptr~~ elem is greater

-2 -1 0 2 3
 ↑ ↑ ↑ ↑
 l l l r

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

20 min

$O(n), O(1)$

$l = 0, r = \text{len}(A) - 1$
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.

IF $A_l^2 \leq A_r^2$

$A_r = A_l^2$

$r--$

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.

1hr 58min

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. $\rightarrow O(n)$ runtime.

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

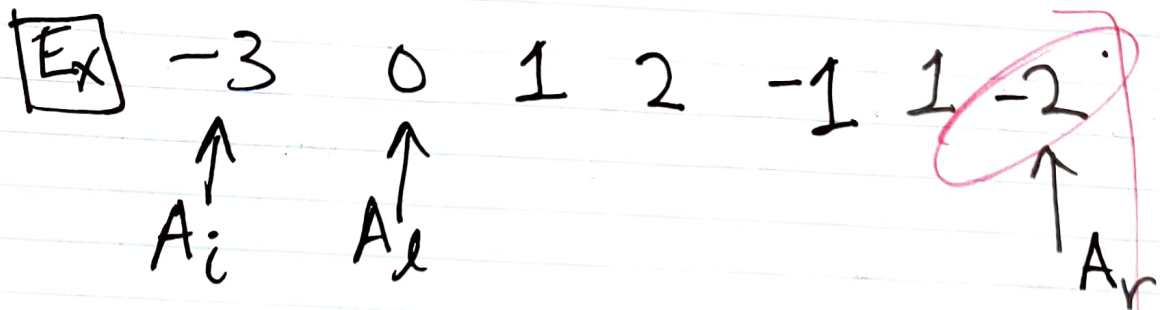
$\uparrow \uparrow \text{---}$

We can take a running sum. ~~and~~

\rightarrow We can take each value in the list and check to find ~~if~~ the ~~value~~ value that ~~we~~ would add to $O(n^2)$ $(-3 + x = 0 \Leftrightarrow x = 3)$
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))

$$-3 + x = 0 \iff x = 3$$



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



This was
the approach
to take

~~For i in range($|A|$)
 $x = \text{abs}(A_i)$~~

We can take two elem at a time
 while traversing the list and find the
 value that would add to 0.

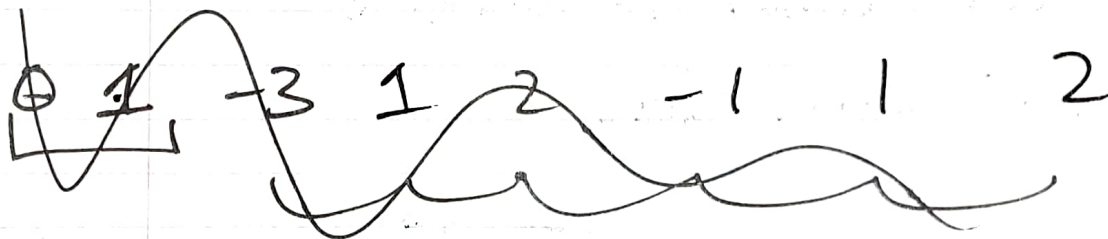
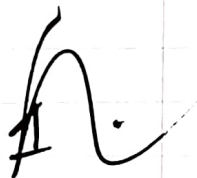
\hookrightarrow Use python in()

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

22 min Quiz

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 $\text{abs}(\text{sum} - \text{target})$
- Need to sort $\rightarrow O(n \log n)$ 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 ~~small~~ sum w/ smallest diff. $\rightarrow O(n^3), O(1)$

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

target = 2

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.

$\rightarrow A_i + A_l + A_r = -2 + 1 + 2 = 1 = \text{curr_sum}$
 $\text{abs}(\text{target} - \text{curr_sum}) = 1 = \text{min_diff}$

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

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

$\text{C.S} = -2$
 $\text{min_diff} = 3$

~~$[-3, -1, 1, 2]$~~

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

~~$\text{C.S} = 2$
 $\text{min_diff} = 1$~~

(*) Need to use while loop inside the for

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

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

$\text{C.S} = 0$
 $\text{min_diff} = 1 \leftarrow$

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

$\text{C.S} = 2$
 $\text{min_diff} = \boxed{1}$

16min

import math

def func(A, t):

A.sort()

min_d = ~~math.inf~~

for i in range(len(A)):

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

while l < r:

c_s = A[i] + A[l] + A[r]

~~c_min~~ d = ~~abs~~ abs(t - c_s)

if ~~min_d~~ min_d > ~~c_s~~ abs(t - c_s)

min_d = c_s

~~l~~ l, r = l + 1, r - 1

while l < r and A_l = A_{l-1}:

~~l~~ l = l + 1

while l < r and A_r = A_{r+1}:

r = r - 1

else:

~~l~~ l = l + 1

return c_s

Umin q_{hit}

~~Umin~~ q_{hit}

Triplet Sum to Zero

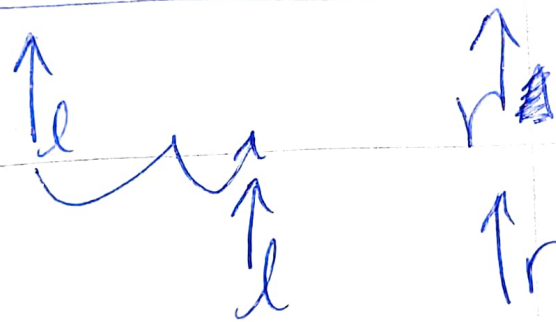
7/20

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

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

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

curr.sum = -3
target.sum = 3



C.S = -2
t.s = 3

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

↑↑ ~~↑~~ End of helper f'n



5/min