Day 4 - §5: Arrays (last time)

6/29/22 Wed

- Go over the problems I solved for the last time and move on to next chapter (§6: Strings) because I'm spending too much time on this chapter.
  - ↳ (5.18) and (5.9) gave most stress.
    - ↳ should I memorize? (The prime problem)

↳ Problems:

(5.1) (5.6) (5.12) (5.18) (5.2) (5.17)

Patterns (From Grokking)

1. Sliding Windows
2. Two Pointers
3. Fast & Slow Pointers
4. Merge Intervals
5. Cyclic Sort
6. In place Reversal of Linked List
7. Tree Breadth First Search
8. Tree Depth First Search
9. Two Heaps
10. Subsets
11. Modified Binary Search
12. Bitwise XOR
13. Top 'K' Element
14. K-way Merge
15. Knapsack
16. Topological Sort

6/29

5.1 Evaluation

- My thinking was corred and figured out the sol'n in time; however, I never looked over Python's Tuple Assignment.

I didn't notice it when I did this problem previously.

So, $A[equal], A[greater] = A[greater], A[equal]$

$\not\Leftrightarrow \begin{cases} A[equal] = A[greater] \\ A[greater] = A[equal] \end{cases}$ NO

instead, it's the same as

$$\begin{cases} temp = A[equal] \\ A[equal] = A[greater] \\ A[greater] = temp \end{cases}$$

(*) This was a very bad mistake and I took too much time figuring trying to it out.

5.6

6/29

Evaluation

- I was trying to use the naive sol'n w/o even knowing it.
- Could've thought about divide and conquer.
- There was no need for the conditionals if I used the python method min()/max()

- This is the algorithm used. Keep track of minimum price thus far.
- If the current price minus the min. price is greater update the max-profit.
- Use a for-loop since we are traversing through the entire list.

44 min.

15.6

- Why didn't I have to use the while-loop, next, current, previous indices for this problem?

- for-loop with i, has a counter runs through the whole ~~list~~ set value
- while-loop repeatedly runs until some condition is no longer true.

- for is better to use when the number of times to repeat is clear in advance.
- while is better when you recognize when to stop

6/29

5.12

- Figured things out in the coding.
- Need to remember random.randint() quicker.
- Could not figure out that I could've had ~~random~~ repeating items if I don't set my params. ~~[illegible]~~ correctly (even though I had the intuition before this) (I had the answer correctly prev.'ly but tried to solve it similar to book this time.)

→ rand = random.randint(i, len(A) - 1)

this way, we don't have duplicates.

↳ Should've done example first.

6/29

5.2 Evaluation

- I had the idea correct but could not execute on the code.
- I have to take of the edge cases and the ~~base~~ end case.

- In the code, since I ~~[illegible]~~ added the 1 on the tenth digit,
  ↳ (ex: 1, 2, 9)

~~I don't [illegible]~~

- I can't start the iteration from 0 (rather at 1)
  ↳ for i in reversed(range(1, len(A)))
- Since we are iterating through a reversed list, we start at len(A)-1 and ~~[illegible]~~ decrement.

6/29

5.1

Input: List and int

Output: List w/ item rearranged.

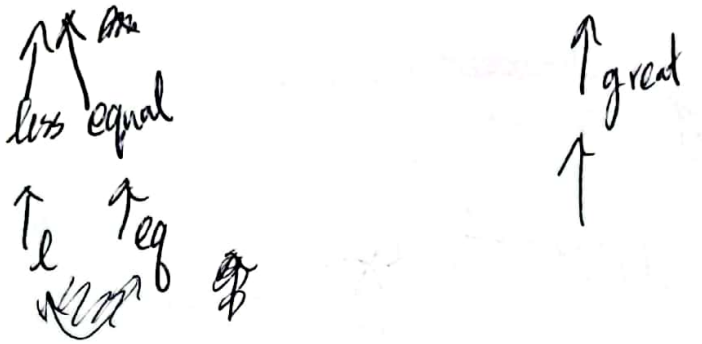
- The one of the input is the pivot index where, all the items need to change position ~~as if~~ if they are less than, equal, or greater.
- The Naive approach is to take an item from the input list and check w/ the item at the pivot index.
- The input List is unsorted.
- What I can do is to create variables for items that are less, equal, or greater where for every items that ~~are~~ mets those conditional I ~~will~~ switch position w/ the current item (then increment/decrement)
- Also, I don't have to change position for items that are equal.

8

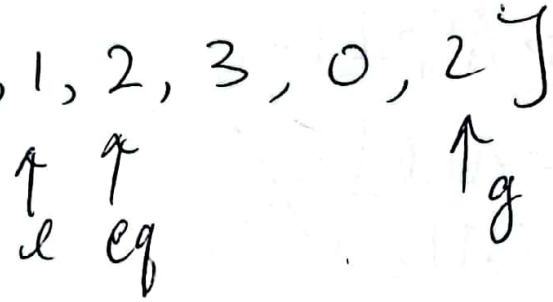
1/3

Ex $p.i = 2$

$A = [1, 0, 2, 3, 0, 2]$ $A[p.i] = 2 > 1$



$\sim [0, 1, 2, 3, 0, 2]$



$A[p.i] = 2 < 3$

$\sim [0, 1, 2, 2, 0, 3]$

$A[p.i] > 0$

$\sim [0, 0, 2, 2, 1, 3] \sim [0, 0, 1, 2, 2, 3]$

2/3

```
def fun(A: List[int], p_i: int) -> List[int]:
    #indices
    l = 0
    e = 0
    g = len(A) - 1
    while e < len(A):
        if A[e] > A[p_i]:
            A[e] = A[l]
            A[l] = A[e]
            l += 1
        if A[e] == A[p_i]:
            e += 1
        else:
            A[e] = A[g]
            A[g] = A[e]
            g -= 1
    return A
```

O(n), O(1)

3/3

6/29

5.6

Input: List[int]
Output: float
↳ ~~Same thing~~ Something to ask to interviewer

Constraints
- We are comparing differences from left to right.
- The differences need not be consecutive / adjacent → Don't use "for i".
- Items in the inputs are positive ↳ For Naive we can.

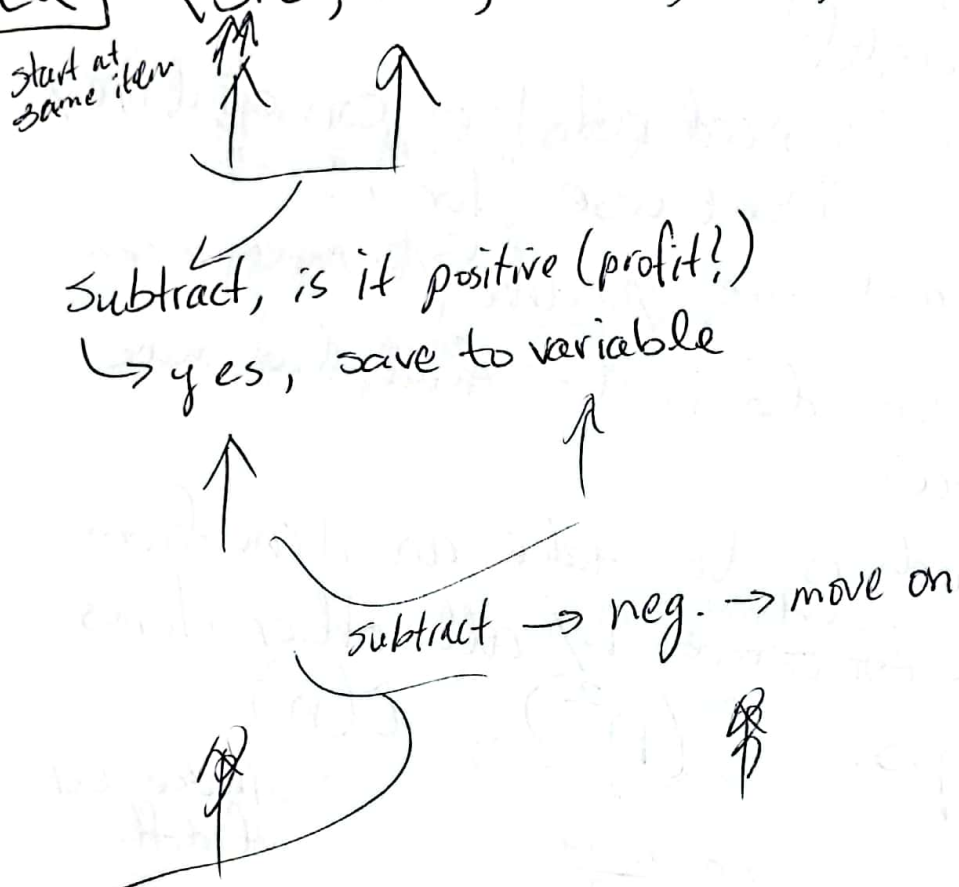
What I want to do is to find the max in the differences.
A Naive Approach is to take an item from the list and ~~compare~~ subtract w/ all other items using 2 loops. $O(n^2)$, $O(n)$
↳ from $\frac{n(n+1)}{2}$ ↳ Make list of diff.
Then append to new list and find the maximum. $O(n \log n)$

1/4

- Do I need to append to new list?
- How can I use pointers?

We can set variables for current, next, prev.

Ex <310, 315, 275, 295, 260, 270, 290, 230>



↳ So we can set up a conditionals to check if the current item is greater than the other items. before doing computation.

13min

2/4

```
def fun(A: List[int]) -> float:
    current = 0
    next = 0
    max = 0
    while current < len(A):
        if A[current] > A[next]:
            next += 1

        else:
            max = A[current] - A[next]
            next += 1
            if next == len(A) - 1
                current += 1
                next = current.

    return max
```

shalld it be >/ ?

X

Redo

```
def fun(prices: List[float]) -> float:
    min_price = float('inf')
    max_profit = 0.0

    for price in prices:
        min_price = min(min_price, price)
        temp = price - min_price
        max_profit = max(max_profit, temp)

    return max_profit
```

4/4

5.12

6/29 1/4/3

Input: List[int] and int
Output: List[int]

- For this problem, we are taking a list and creating another or truncating the input list that contains the input size amount of elements.

Constraints: • The input list contains unique items.
• Edge cases ~ Empty list, size must be nonneg and ℤ.

The Naive Approach is to use the Python rand() method and append to a new list. → O(s) space where s is the input size
↳ Instead, we can also delete from the existing input list.
↳ Deleting from list takes O(1) per operation → O(n-s) where n is size of the list.

~~However~~,

Furthermore, we can expand on the Naive Approach and pick a random position on the list and and rearrange the input list S number of times. This will give us the list w/ all the random items with the correct ~~number~~ cardinality.

The random number rand must be a number between 0 and s

$0 \leq rand < S$, $rand \in \mathbb{Z}$

* rand is an index

Jumping straight into sol'n. 3/3

```
def fun(A: List[int], s: int) -> List[int]:
    rand = random number between 0 & s.
           random.randint(0, s) ?
    for i in range(s):
        A[i] = A[rand]

    return A
```

16mm

5.2

Input: List[int]

Output: ↗

Constraints: $\forall e \in A$, $e \in [0, 9]$, $e \in \mathbb{Z}$

- Naive Approach is to convert the items in the input to string join them and convert to integer and add 1 then split them.
- Strings are immutable so don't know exactly how much it will take but it will take $O(n)$ space, where n is the length of input
  - $O(n+1)$ "worst case"

- A better approach is to use pointers and conditionals.
- We can iterate through the input list and check to see if it equals 10 after adding one.

Ex

1, 2, 9 $\xrightarrow{D+1}$ 1, 3, 10

add 1 here

0 1 2

if 10, then make this decimal equal 0 and add 1 to next decimal.

⇒ "Edge Case" if the number has all 9's, then we need to append a 1 to the begginning of the list.

Ex 9, 9, 9 ~ 9, 9, 10 ~ 9, 10, 0

↳ 1, 0, 0, 0

2/3

```
def fun(A: List[int]) -> List[int]:
    # Add the 1 to end of list
    A[-1] += 1
```

Should I reverse the list?

```
    for i in reversed(range(len(A))):
        if A[i] == 10:        else: break
            A[i] = 0          Since reversed.
            A[i-1] += 1
            if A[0] == 0 and i == 0:
                A[i] = 1
                A.append(0)
    return A.
```

O(n), O(1)

15 min

3/3