Striver A-Z

Outline

1. Strings

a.Easy b.medium

2. Arrays

a.Easy b.medium c. Hard

3.Binary Search

a.Easy b.medium c. Hard

check these out

Police & theif

Porochut of array except self → also no ÷ version too

Trapping Rain water

Data Structures

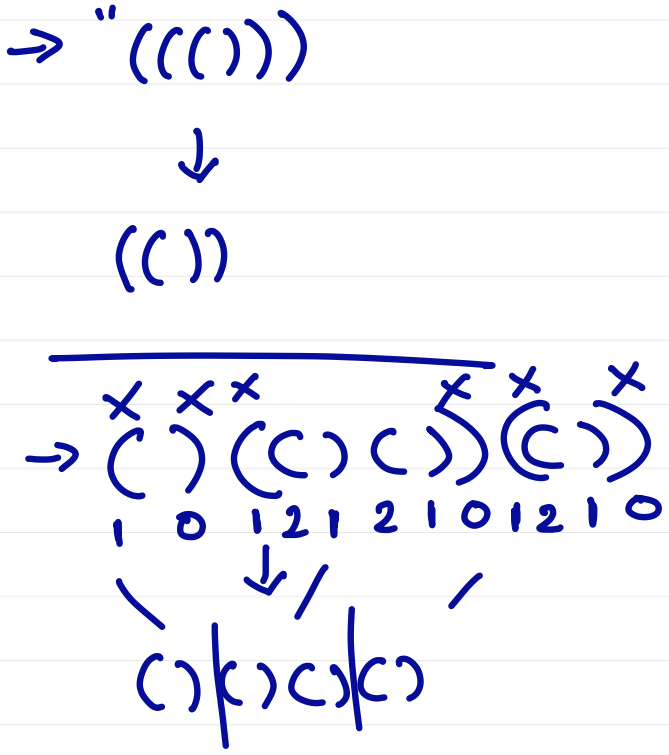
Strings

Easy

1) Remove Outermost parenthesis

```python
def removeouterparanthesis(s):
    count=0
    res=""
    for i in range(len(s)):
        if s[i] =="(":
            if count>0:
                res+=s[i]
            count+=1
        else:
            count-=1
            if count>0:
                res+=s[i]
    return res

s=input()
print(removeouterparanthesis(s))
```



Logic

1) → if open brace & count>0 → Add to res.
if count == 0 (starting of outer most para sequence)
count++

2) → close brace → count--
after.
count > 0 → Add to res.
count == 0 → –

2) Reverse words in a String

```python
s=input()
mylist=list(s.split())
rev=""
rev=" ".join(mylist[::-1])
# for x in mylist:
#     rev=str(x)+" "+rev
print(rev)

#variable initialisation
#Building step by step? → ✅ Initialize first.
#Assigning in one go? → ❌ No need to initialize.
```

→ no need of list(s.split())

↓

s.split() → list.

Logic:

s="welcome to the jungle" (words-reverse)

↓

mylist = list(s.split()) → ["welcome", "to", "the", "jungle"]

" ".join(mylist[::-1]) → jungle the to welcome

---

3) Largest odd number in a string (i/p)

```python
s=input()
s=s.lstrip('0')
for i in range(len(s)-1,-1,-1):
    if int(s[i])%2!=0:
        print(s[0:i+1])
        break
```

→ 0 2 3 4 4 5 6 6

↓ removes left zeroes.

↓ odd nos.

∴ → s[0:i+1]

↓

odd number index.

4) Longest common Prefix

```python
def longestCommonPrefix(strs):
        prefix=""
        for i in range(len(strs[0])):
            char=strs[0][i]
            for word in strs:
                if i>=len(word) or char!=word[i]:
                    return prefix
            prefix+=char
        return prefix
strs=list(input().split())
print(longestCommonPrefix(strs))
```

['flower', 'fly', 'flow']

↓

'fl' → prefix.

Brute-force:

→ take 1st ch of 1st word and cmp with other words.

(also check for when i >= len(word))

---

5) Isomorphic String (If ch in one str can be replaced to get another str)

```python
def isomorphic(s,t):
    mydict={}
    for i,j in zip(s,t):
        if i in mydict:
            if mydict[i]!=j:
                return False
        else:
            mydict[i]=j
    return len(set(mydict.values()))==len(mydict)
### for the case of s=badc and t=bada where a and c map to the same value a which contradicts the stmt
s=input()
t=input()
print(isomorphic(s,t))
```

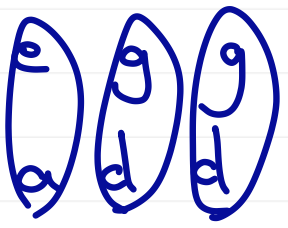
needed in case → when 2 or more values are equal.

i/p

s = "egg"

t = "add"

→ true.

→ 

→ dict = e : a (add item) → not there in dict.
g : d (add item)

→ g... check if it's value == j

☆ Edge-case: badc / bada →

b : b
a : a
d : d
c : a

} all are valid but these values are same.

6) Check if a String is rotation of another string

```python
def rotatestring(s,goal):
    if len(s)!=len(goal):
        return False
    return True if s in goal+goal else False

s=input()
goal=input()
print(rotatestring(s,goal))
```

→ i/p

s = abcde

goal = cdeab

Brute-force: try all slicing combinations $O(n^2)$

Optimal: check if s in goal + goal.

goal + goal = cdeabcdeab $O(2n)$ → len(goal)

↓

my string s

---

7) Check if 2 sthings are anagrams to each other

```python
from collections import Counter
def anagram(s,t):
    return Counter(s)==Counter(t)
```

anagrams

↓

If 2 strings are the same word but with diff letters arrangement

Brute-Force:

↳ Sort & compare

T: (O(nlogn))

Optimal-Approach:

↳ Compare their frequency Counter's (dicts)

T: O(n)

Medium

1) Sort characters by frequency

```python
from collections import Counter
def frequencySort(s):
    freq=Counter(s)
    sorteddict=dict(sorted(freq.items(), key=lambda item: (-item[1],item[0])))
    s=""
    for i in sorteddict.keys():
        s=s+(i)*sorteddict[i]
    return s

s=input()
print(frequencySort(s))
```

o/p

→ s = abbccc

↓

s = cccbba

logic :- We create a counter to get freq's.

countername.

sorteddict = dict (sorted (freq.items(), key = lambda x : (-x[1], x[0])))

(or)

sorteddict = dict (sorted (freq.items(), key = lambda x : x[1], reverse = True))

{c:3, b:2, a:1}

↳ print using multiplicity.

---

2) Maximum Nesting Depth.

```python
def maxDepth(s):
        count=0
        max=0
        for i in s:
            if i=="(":
                count+=1
                if count>max:
                    max=count
            elif i==")":
                count-=1
        return max
s=input()
print(maxDepth(s))
```

→ i/p : "(1+(2+3)+((8)/4))+1" = 3

(3 open brace)

Just use a loop & using if & else update count & max.

3) Roman to Integer

```python
def romanToInt(s):
        roman={"I":1,"V":5,"X":10,"L":50,"C":100,"D":500,"M":1000}
        value=0
        i=len(s)-1
        while(i>=0):
            if i==0:
                value+=roman[s[0]]
                i-=1
            elif roman[s[i]]>roman[s[i-1]]:
                value+=roman[s[i]]-roman[s[i-1]]
                i-=2
            else:
                value+=roman[s[i]]
                i-=1
        return value
s=input()
print(romanToInt(s))
```

i/p

→ LVIII

↓

58

→ MCMXCIV

↓

1994.

M | CM | XC | IV

1000 900 90 4

logic :-

Start from back. (MCMXCIV)

if r[s[i]] > r[s[i-1]] (i)

↓

value += r[s[i]] - r[s[i-1]]

5 - 1 = 4.

↓ decrement i → 2 times.

Edge case:

MCMXCIV

↑

i=0 → we can't have i=-1 Hence we write a seperate condition before i & i-1 check.

4) Implement Atoi ☆

```python
def myAtoi(s):
        sign=1
        ans=""
        flag=0
        for i in range(len(s)):
            if s[i]==" " and flag==0:
                continue
            if s[i]=="-" and flag==0:
                sign=-1
                flag=1
                continue
            if s[i]=="+" and flag==0:
                flag=1
                continue
            if s[i].isdigit():
                ans+=s[i]
                flag=1
            else:
                break
        if not ans:
            return 0
        ans=int(ans)*sign
        INT_MIN=-2**31
        INT_MAX=2**31-1
        if ans>INT_MAX:
            return INT_MAX
        if ans<INT_MIN:
            return INT_MIN
        return ans
s=input()
print(myAtoi(s))
```

Logic:- → use continue

1) ignore " ",
Remember "-" (neg sign)
stop if "+", "-" comes
↳ but if they come at the start ignore.

So i am using flag cuz " ", "-", "+" are only allowed before first digit
so when i encounter
↓
set flag = 1

i/p = " -042" → -42
= "1337c0d3" → 1337.

[Mul ans with sign if neg is there]

→ **Rounding:** INT_MIN = -2**31
INT_MAX = 2**31-1

if ans > INT_MAX → INT_MAX
ans < INT_MIN → INT_MIN
} Return these

5) Count no of substrings (non-distinct)

```python
def countsubstrings(n):
    return (n*(n+1))//2
s=input()
print(countsubstrings(len(s)))
```

→ just a sum approach

abcde → 5 → n

= 5 = 2 - 1

15 → ans.

(not important Q)

---

6) Longest Palindrome Substring

```python
def helper(res, reslen, s, l, r):
    while l>=0 and r<len(s) and s[l]==s[r]:
        if r-l+1>reslen:
            res=s[l:r+1]
            reslen=r-l+1
        l-=1
        r+=1
    return res,reslen
def longest_reslen_substring(s):
    res=""
    reslen=0
    for i in range(len(s)):
        res,reslen=helper(res,reslen,s,l=i, r=i)
        res,reslen=helper(res,reslen,s,l=i, r=i+1)
    return res

s=input()
```

i/p

→ s = "abbaba"

↓

s = abba

o/p.

palindrome < odd → ababa
even → abba.

Logic: odd → a b a b a (start at i → len(s))

↑ (l=r=i)

l, r

↓

l-- & r++

even → a b a b a (l=i, r=i+1)

↑ ↑

l r

& constantly check (update lengths) for palindrome

7) Sum of Beauty of all Substrings

Beauty of a Substring

beauty = (max freq - min freq)

```python
from collections import Counter
def sumofbeauty(s):
    total=0
    beauty=0
    for i in range(len(s)):
        freq=Counter()
        for j in range(i, len(s)):
            freq[s[j]] += 1
            beauty=max(freq.values()) - min(freq.values())
            total+=beauty
    return total
s=input()
print(sumofbeauty(s))
```

i/p

-> s = aabcb

o/p : 5

(aab, aabc, aabcb, abcb, bcb)

have non-zero beauties.

Logic :-

aabcb -> check all substring & get freq - array

do beauty = (max freq - min freq)

↳ add to total.

☆ (Also try using fixed array = [0]×26)

↳ Counter time cost > fixed array time cost.

(overhead time is more)

Arrays

Easy

1) Largest element in an array

```python
arr=list(map(int,input().split()))
max_ele=arr[0]
for i in range(1,len(arr)):
    if arr[i]>max_ele:
        max_ele=arr[i]
print(max_ele)
```

(or use max(array))

---

2) Second largest element in an array without sorting

```python
arr=list(map(int,input().split()))
max_ele=float('-inf')
second_max_ele=float('-inf')
for i in range(len(arr)):
    if arr[i] > max_ele:
        second_max_ele = max_ele
        max_ele = arr[i]
print(second_max_ele)
```

---

3) Check if array is sorted

```python
arr=list(map(int,input().split()))
flag=True
for i in range(1,len(arr)):
    if arr[i]<arr[i-1]:
        flag=False
        break
print(flag)
```

---

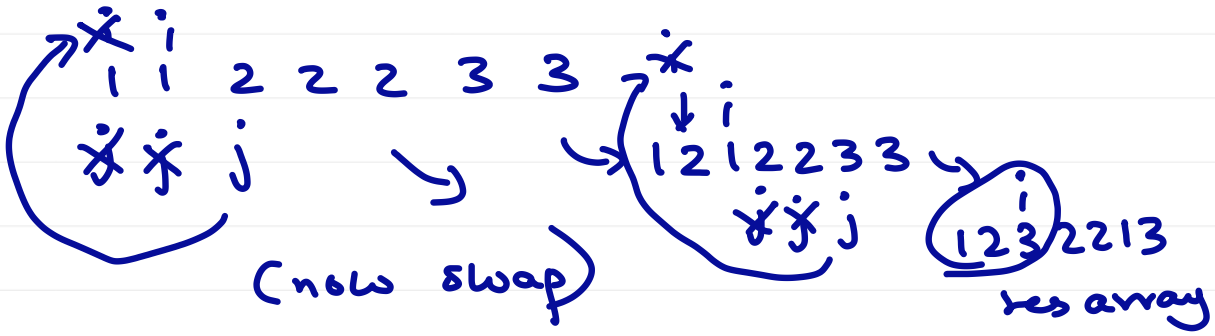
4) Remove duplicates from Sorted array

```python
def remove_duplicates_from_sorted_array(arr):
    i=0
    for j in range(len(arr)):
        if arr[j]!=arr[i]:
            i+=1
            arr[i]=arr[j]
    return arr[:i+1]

```

→ i/p:
arr: [1,1,2,2,2,3,3]
↓
o/p: [1,2,3]

logic:



☆ Don't be confused where when they are not equal → i is incremented first & then SWAP.

---

5) Left rotate array by one place

```python
def left_rotate_array_by_one_place(arr):
    temp=arr[0]
    for i in range(len(arr)-1):
        arr[i]=arr[i+1]
    arr[-1]=temp
    return arr
arr=list(map(int,input().split()))
print(left_rotate_array_by_one_place(arr))
```

---

6) left rotate array by k places

s[:k] = ≡≡≡

slicing assignment.

```python
def rotate_array_by_k_elements(arr, k):
    k=k%len(arr)
    arr.reverse()
    arr[:k] = reversed(arr[:k])  # Revers
    arr[k:] = reversed(arr[k:]) # Reverse
    return arr
```

} can't use reverse() cuz slicing gives a copy.

7) Move zeroes to end

```python
def move_zeroes_to_end(arr):
    j=0
    for i in range(len(arr)):
        if arr[i]!=0:
            arr[j]=arr[i]
            j+=1
    for i in range(j,len(arr)):
        arr[i]=0

arr = list(map(int, input().split()))
move_zeroes_to_end(arr)
print(arr)
```

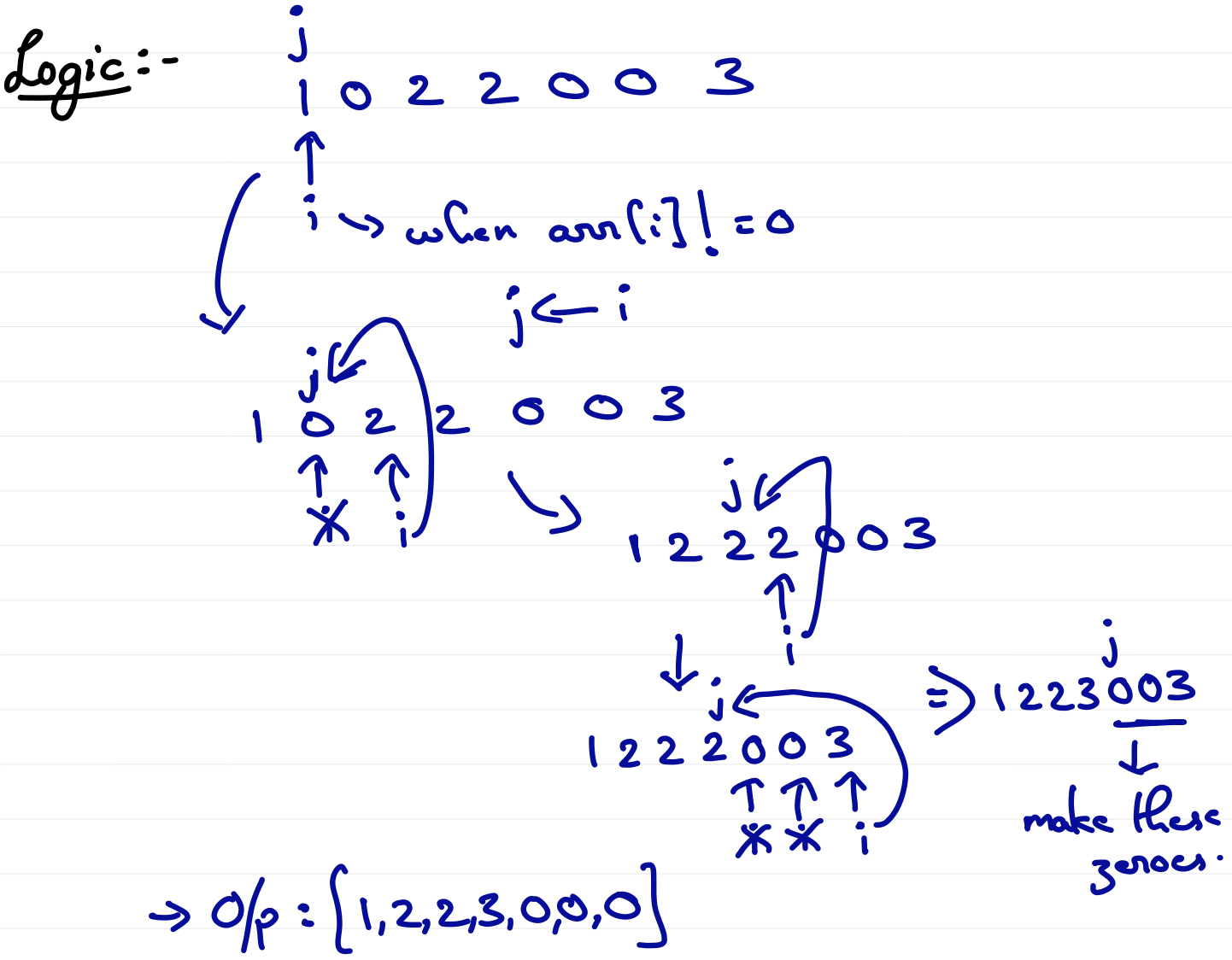
i/p

→ arr = [1,0,2,2,0,0,3]

⇓

o/p : 1223000

Logic :-



→ o/p : [1,2,2,3,0,0,0]

---

8) Linear Search

```python
def linearsearch(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return f"found at index {i}"
    return "not found"
arr = list(map(int, input().split()))
target = int(input())
print(linearsearch(arr, target))
```

→ O(n)

---

9) Union of two sorted arrays

```python
def union_of_two_sorted_arrays(arr1, arr2):
    i,j=0,0
    res=[]
    while i<len(arr1) and j<len(arr2):
        if arr1[i]<arr2[j]:
            if arr1[i] not in res:
#check is there cuz in union we don't want duplicates
                res.append(arr1[i])
#and this is used only for when an array has its own dupliactes
            i+=1
        elif arr1[i]>arr2[j]:
            if arr2[j] not in res:
                res.append(arr2[j])
            j+=1
        else:
            if arr1[i] not in res:   # when both are equal we only need to consider one and increment both i and j
                res.append(arr1[i])
            i+=1
            j+=1
    while i<len(arr1):
        if arr1[i] not in res:
            res.append(arr1[i])
        i+=1
    while j<len(arr2):
        if arr2[j] not in res:
            res.append(arr2[j])
        j+=1
    return res

```

} Iterating over left over ele's

I/P

arr1 = [1,2,2,4]

arr2 = [1,3,5,7]

O/P

↓

[1,2,3,4,5,7]

Union of A & B = Call distinct elements in A & B)

Logic:- ~ Merge Sorted arrays

But, 1st → if arr1[i] not in res: (also do for arr2[i])

1

Check this cuz ↓ we don't want repitive element (which are from same array)

arr1 = [1, 2, 2, 3]
↑ not added.

2nd → else:
if arr1[i] not in res:
res.append(arr1[i])
i+=1
j+=1

this is for when arr1[i] == arr2[j]
(we only add arr1[i] & increment i,j

3rd → Don't forget to iterate over remaining set of Elements when i or j > len(arr)

---

10) Find missing number in an array

ip
↓
arr = [0,1,2,4]

o/p: 3

```python
def find_missing_number_in_array(arr):
    n = len(arr)   # Since one number is missing
    total_sum = n * (n + 1) // 2
    array_sum = sum(arr)
    return total_sum - array_sum
arr = list(map(int, input().split()))
print(find_missing_number_in_array(arr))
```

↳ Using Sum of n terms formulae.

⑩

```python
def find_missing_number_xor(arr):
    n = len(arr)
    xor_all = 0
    xor_arr = 0
    for i in range(n+1):
        xor_all ^= i
    for num in arr:
        xor_arr ^= num
    return xor_all ^ xor_arr

arr = list(map(int, input().split()))
print(find_missing_number_xor(arr))
```

→ In XOR

a^0 = a

a^a = 0

So

xor-arr = [0,1,2,4]

xor_all = [0,1,2,3,4]

↳ Using XOR Approach.

XOR → 3 (unique ele)

---

⑪ Maximum Consecutive ones

```python
def maximum_no_of_ones(arr):
    count = 0
    max_count = 0
    for num in arr:
        if num==1:
            count+=1
            max_count = max(max_count, count)
        else:
            count = 0
    return max_count
```

→ O(n) approach

↓

Just loop through.

---

⑫ Find the number that appears once (others appear twice)

```python
def find_which_no_appear_once(arr):
    xor=0
    for num in arr:
        xor^=num
    return xor
```

→ [1,1,2,3,3,4,4]

0 2 0 0

2 (XOR)

↓

for same ele = 0

---

13) Longest Subarray with given sum K (positives)

```python
def longestsubarraywithsumK(arr):
    j=0
    sum=0
    max_length=0
    for i in range(len(arr)):
        sum+=arr[i]
        while sum > k:
            sum -= arr[j]
            j += 1
        if sum == k:
            max_length = max(max_length, i - j + 1)
    return max_length
arr=list(map(int, input().split()))
k = int(input())
print(longestsubarraywithsumK(arr))
```

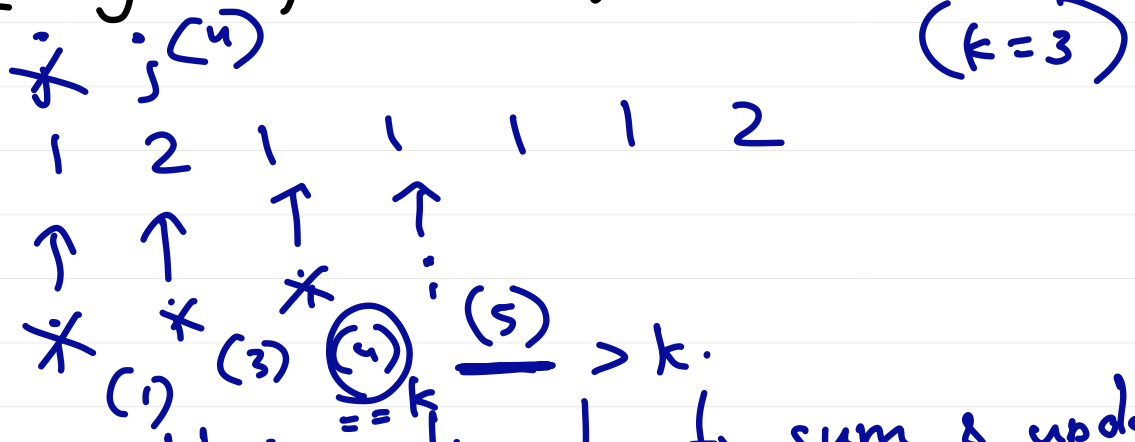
⇒ i/p

arr: 1 2 1 1 1 1 2

↓

o/p: 4

Logic (using 2-pointers approach)



I will add i continously to sum & update max_len. when (sum == k)

when sum > k i will increment j until (sum ! > k) → while (sum > k)

sum -= arr[j]

j += 1

14) Longest Subarray with given sum K (+ve & -ve)

```python
def longest_subbary_with_sum_k(arr,k):
        prefix_sum=0
        seen={}
        max_len=0
        for i in range(len(arr)):
           prefix_sum+=arr[i]
           if prefix_sum==k:
                   max_len=max(max_len,i+1)
           if prefix_sum-k in seen:
                 max_len=max(max_len,i-seen[prefix_sum-k])
           if prefix_sum not in seen:
                 seen[prefix_sum]=i
        return max_len
```

→ i/p

arr = [1, 2, -1, 2, -1, 4]

k = 5

o/p = 6

Logic (Prefix-sum + hash map)

2-pointer approach ✗ (doesn't work)

↳ cuz if you remove element's it doesn't guarentee that sum is present or not.

(1, -1, 5, -2) for k=3

↳ it doesn't count.

Add arr[i] to prefix_sum.

if prefix_sum = k → update max_len.

if prefix_sum-k is in seen:

↳ then we update max_len (accordingly)

if prefix_sum not in seen

↳ add prefix_sum & i to seen

Medium

1) 2-Sum Problem

```python
def two_sum(arr,k):
    seen = {}
    for i in range(len(arr)):
        if k-arr[i] in seen:
            return [seen[k-arr[i]],i]
        seen[arr[i]]=i

arr = list(map(int, input().split()))
k=int(input())
print(two_sum(arr,k))
```

I/p

arr: 2 7 11 15

k: 9

o/p: [0,1]

↓

indexes

Logic:-

just for every element check if k-num exists in seen if yes return the indexes.

---

2) Sort colors

a)

```python
def sortColors(nums):
    colors = [0, 0, 0]
    for i in nums:
        if i==0:
            colors[0]+=1
        elif i==1:
            colors[1]+=1
        else:
            colors[2]+=1
    k=0
    for i in range(len(nums)):
        while colors[k]==0:
            k+=1
        nums[i]=k
        colors[k]-=1
    return nums
```

→ I/p

arr: 0 1 0 0 2 2 1

o/p: 0 0 0 1 1 2 2

→ just count 0's, 1's & 2's and change the values in array until all counts are 0.

we use while because in case of [2,2] → counts: 0 0 2

0 1 2

(or)

```python
def sortColors(nums):
    count0 = nums.count(0)
    count1 = nums.count(1)
    count2 = nums.count(2)
    nums[:] = [0]*count0 + [1]*count1 + [2]*count2
    return nums
```

if counts[0] == 0 then we immediately add next count ✗

b) Using 3-pointer Approach

```python
def sort_colors_2(arr):
    l=0
    m=0
    h=len(arr)-1
    while m<=h:
        if arr[m]==0:
            arr[m],arr[l]=arr[l],arr[m]
            m+=1
            l+=1
        elif arr[m]==2:
            arr[m],arr[h]=arr[h],arr[m]
            h-=1
        else:
            m+=1
    return arr
```

i/p

arr: 0 1 0 0 2 2 1

o/p : 0 0 0 1 1 2 2

Dutch National Flag Voting Algorithm

Logic:

3-pointers → l, m, h

l → 0's, m → 1's, h → 2's.

l → 0, m → 0, h → len(arr)-1

swap

0 1 0 0 2 2 1

0 0 1 0 2 2 1

swap

0 0 0 1 2 2 1

swap

0 0 0 1 1 2 2 (stop)

☆ when arr[m] == 2

we don't change both m & h

coz if m & h both are 2

- - 2 - - 2 → - - 2 2 (left out)
  m h m h

3) majority element (n/2) times

```python
def majority_element(nums):
    count=0
    value=None
    for num in nums:
        if count==0:
            value=num
            count=1
        elif num==value:
            count+=1
        else:
            count-=1
    return value
```

i/p

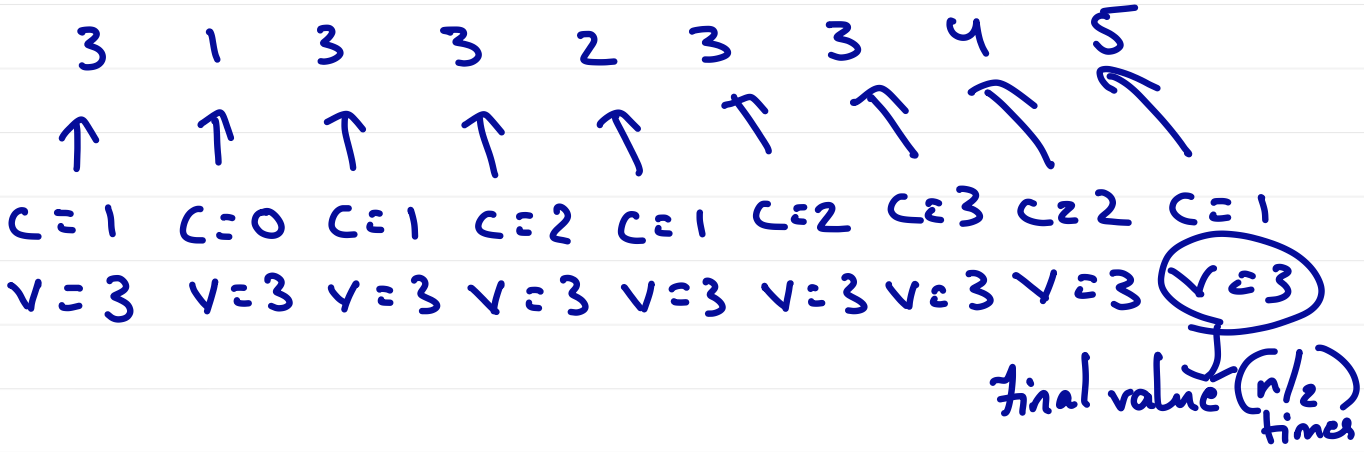
nums = 3 1 3 3 2 3 3 4 5

o/p : 2

Booye-moore Majority Voting algorithm

Logic:- Using count & value.

c = 0
v = None



4) Maximum Subarray sum

```python
def maximum_subarray_sum(arr):
    curr_sum=0
    max_sum=float('-inf')
    for num in arr:
        curr_sum=max(num, curr_sum + num)
        max_sum=max(max_sum, curr_sum)
    return max_sum
arr=list(map(int, input().split()))
print(maximum_subarray_sum(arr))
```

I/p

arr: -2 1 -3 4 -1 2 1 -5 4

o/p: 6

Kadane's Algorithm

↓ O(n)

Logic:

In this algo we have curr_sum & max_sum
↓ 0 ↓ -∞

	-2	1	-3	4	-1	2	1	-5	4
c:	-2	1	-2	4	3	5	6	1	5
M:	-2	1	1	4	4	5	6	6	6 → Ans.

Here main logic → If arr[i] > curr_sum + arr[i]
then curr_sum = arr[i]
→ we change curr_sum to arr[i] cuz a new subarray starts from there. (before values are not needed)

---

5) Maximum Subarray Sum & Subarray

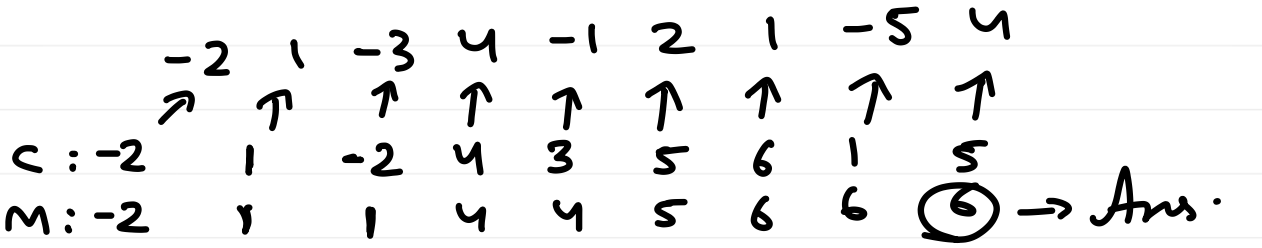
```python
def maximum_subarray_sum_with_subarray(arr):
    start=end=start_temp=0
    curr_sum=0
    max_sum=float('-inf')
    for i in range(len(arr)):
        if arr[i]>curr_sum+arr[i]:
            curr_sum=arr[i]
            start_temp=i
        else:
            curr_sum+=arr[i]
        if curr_sum>max_sum:
            max_sum=curr_sum
            end=i
            start=start_temp
    return [max_sum, arr[start:end+1]]
```

I/p
→ arr: -2 1 -3 4 -1 2 1 -5 4

O/p: 6 [4 -1 2 1]
↓ sum ↓ subarray

Kadane's Algorithm

Logic:- Here, we just need to identify & store the start & end indexes of the subarray which gives us the max_sum.



For start_temp index:

↳ when my arr[i] > curr_sum + arr[i]

then from that index my new subarray starts, start_temp = i (The reason we use start_temp = i cuz not every subarray change will give us a max_sum)

For end index:

↳ we make end = i whenever my curr_sum > max_sum (which tells us including the curr ele increases the max sum)

also, start = temp_start.

---

6) Stock Buy & sell

```
1  def Stocks_buy_and_sell(arr):
2      j=0
3      max_profit=0
4      for i in range(len(arr)):
5          if arr[i]-arr[j] > max_profit:
6              max_profit=arr[i]-arr[j]
7          if arr[i]<arr[j]:
8              j=i
9      return max_profit
10 arr=list(map(int, input().split()))
11 print(Stocks_buy_and_sell(arr))
```

I/p

arr: 7 1 5 3 6 4

O/p: 5

Logic: (Simple)

Use a for loop and go through the array with i & j where i → S.P , j → B.P

Calculate max-profit as well and when we obtain an i which is < j then update j

---

7) Rearrange the array in alternative +&- items

```
def rearrange_ele_by_sign(arr):
    pos=neg=[]
    for num in arr:
        if num>0:
            pos.append(num)
        else:
            neg.append(num)
    for i in range(0,len(pos),2):
        arr[i]=pos[i//2]
    for i in range(1,len(neg),2):
        arr[i]=neg[i-1//2]
    return arr
```

→ basic logic.

just get 2 arrays

pos → positive.

neg → negative.

now, reassign values alternately

---

8) Next-Permutation.

```
1  def next_permutation(arr):
2      n=len(arr)
3      i=n-2
4      while i>=0 and arr[i]>=arr[i+1]:
5          i-=1
6      if i>=0:
7          j=n-1
8          while arr[j]<=arr[i]:
9              j-=1
10         arr[i],arr[j]=arr[j],arr[i]
11     arr[i+1:]=list(reversed(arr[i+1:]))
12     return arr
```

i/p: 4 3 2 5 3 1

o/p: 4 3 3 1 2 5

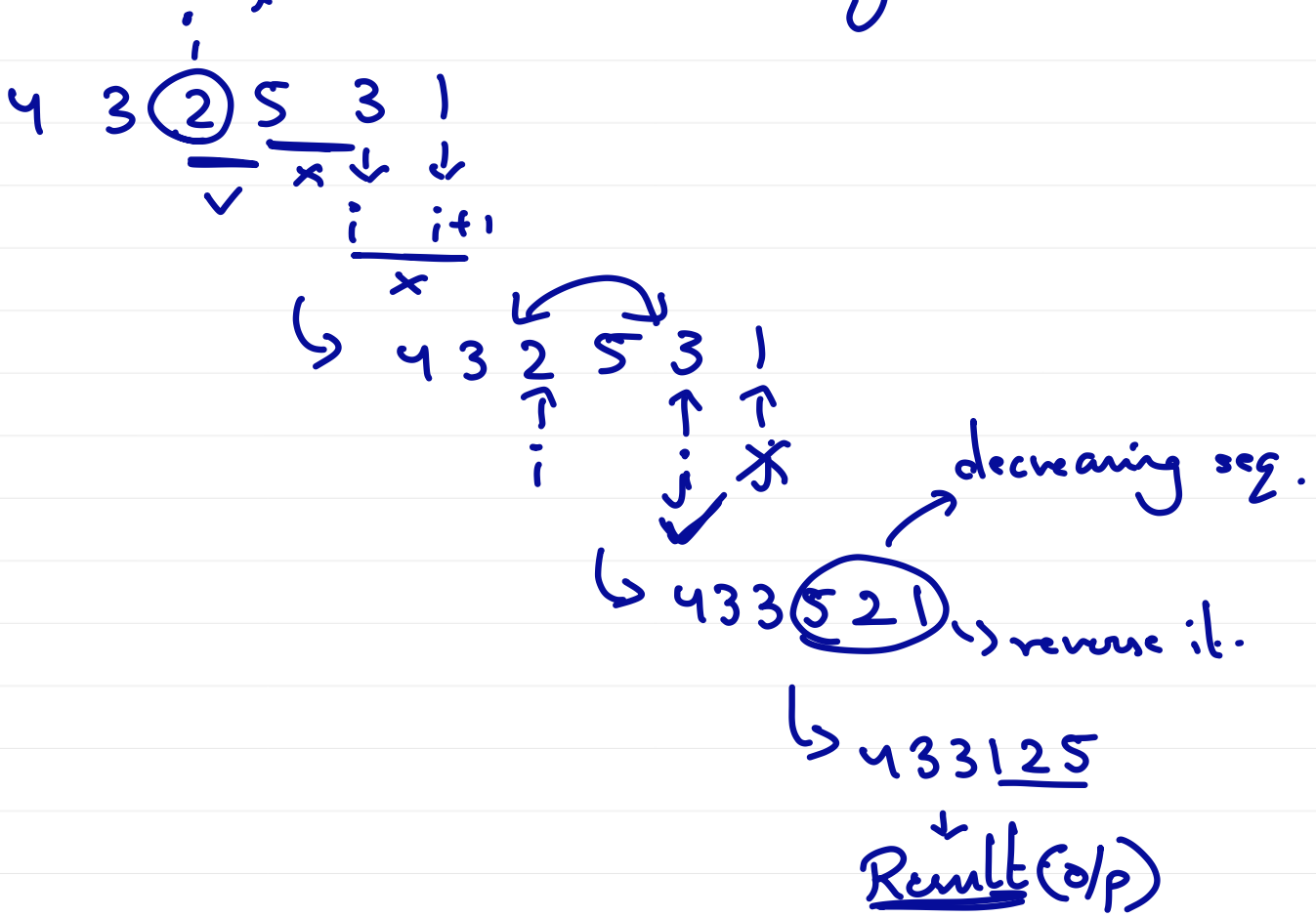
(next permutation → next lexicographical sequence)

Logic:: (3 main steps) → st from right.

→ Identify which element → creates a dip or makes it decreasing (i)

→ Identify the next greater elem than pivot

⇒ swap the two & reverse the right part from (pivot+1) & (i+1) → to get the ans.



---

9) Leaders in array

```python
def leaders_in_array(arr):
    max_val=arr[len(arr)-1]
    arr[len(arr)-1]=1
    for i in range(len(arr)-2,-1,-1):
        if arr[i]>max_val:
            max_val=arr[i]
            arr[i]=1
        else:
            arr[i]=0
    return arr
```

i/p : 16 17 4 3 5 2

o/p : 0 1 0 0 1 1

leader → the ele should be greater than all ele on right side.

Logic: (Simple)

Start from right & keep a max_val & if an ele is greater than max_val → 1 else 0
↓
also update max_val.

10) Longest Consecutive sequence in an array

```python
def longest_consecutive_sequence(arr):
    myset=set(arr)
    max_len=0
    start=None
    for num in arr:
        curr=num
        while curr in myset:
            curr+=1
        length=curr-num
        if length>max_len:
            max_len=length
            start=num
    return [start+i for i in range(max_len)]
```

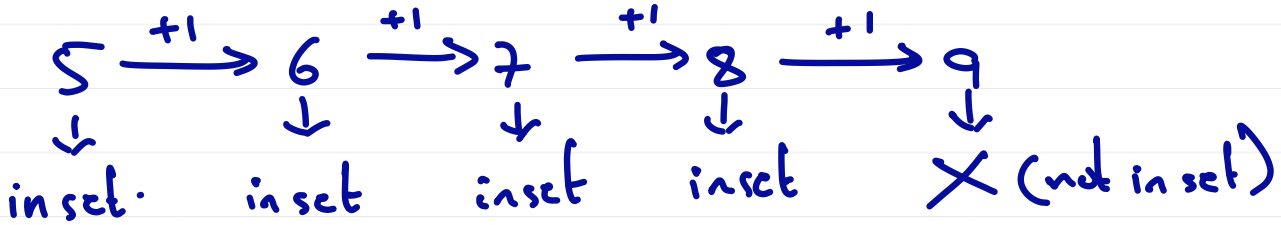
i/p: 3 8 5 7 6

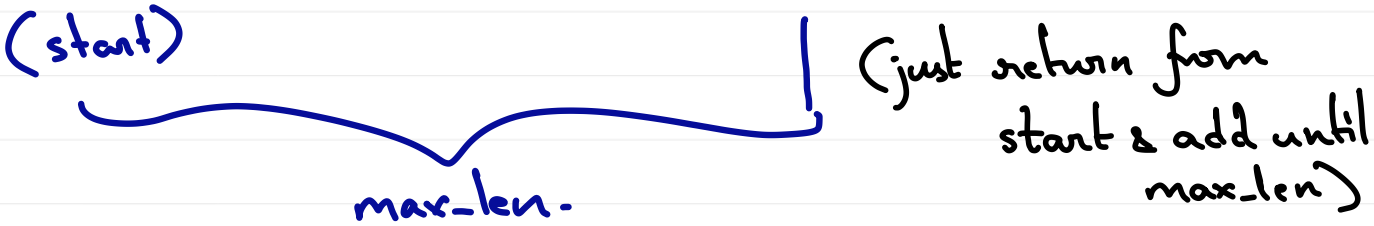
o/p: [5, 6, 7, 8]

logic:

we use a set to have O(1) look up.

(Simple) → like if we see 5 go to 5 for example.





11) Set Matrix zeroes.

matrix - input:

```
n, m = map(int, input().split())
matrix = [list(map(int, input().split())) for _ in range(n)]
print(matrix)
```

3 3
1 2 3
4 5 6
7 8 9

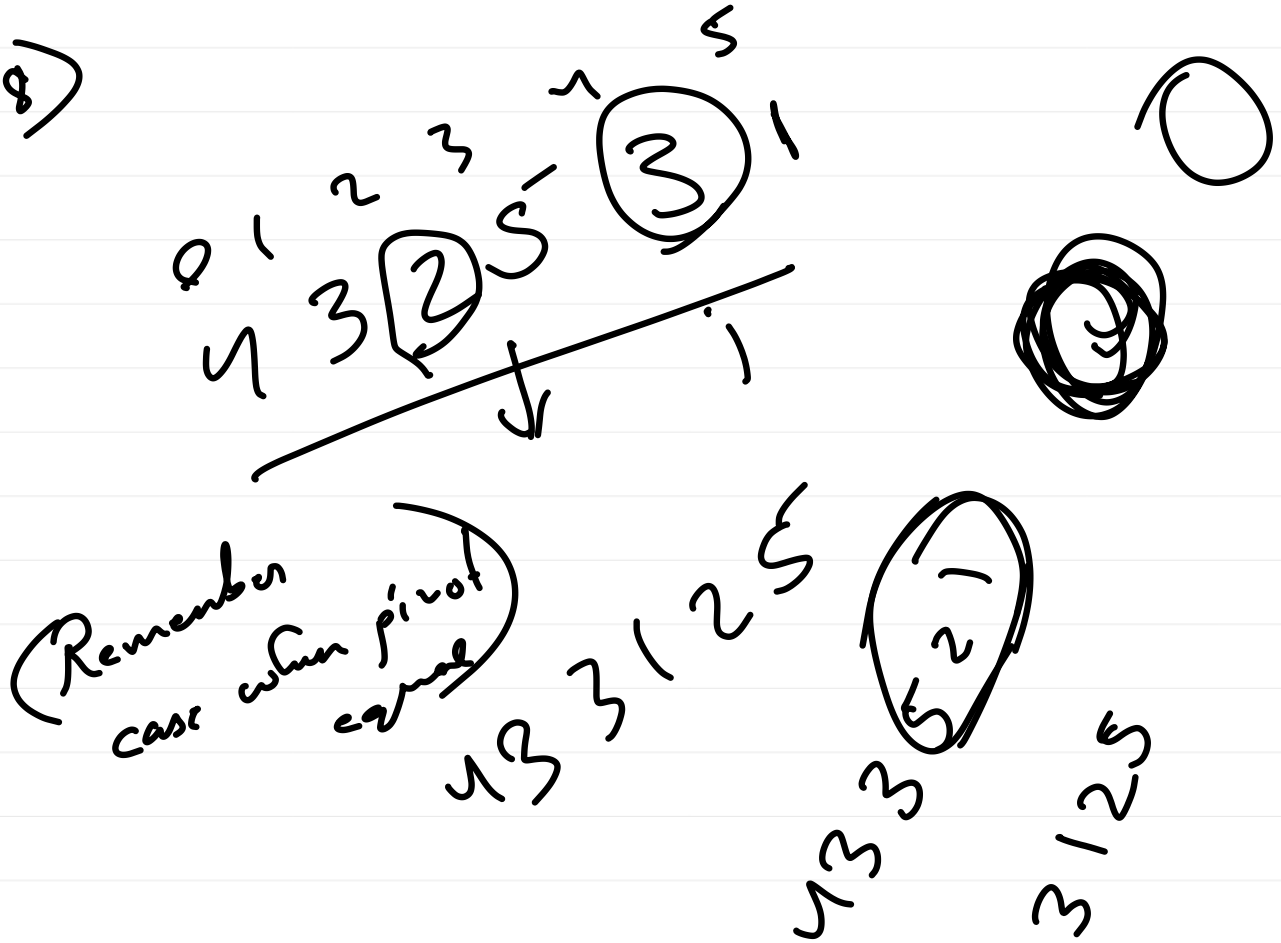
→ [[1,2,3], [4,5,6], [7,8,9]]

```python
def set_matrix_zeroes(matrix,n,m):
    first_row=False
    first_col=False
    for j in range(m):
        if matrix[j][0]==0:
            first_row=True
            break
    for i in range(n):
        if matrix[0][i]==0:
            first_col=True
            break
    for i in range(1,n):
        for j in range(1,m):
            if matrix[i][j]==0:
                matrix[i][0]=0
                matrix[0][j]=0
    for i in range(1,n):
        for j in range(1,m):
            if matrix[i][0]==0 or matrix[0][j]==0:
                matrix[i][j]=0
    if first_row:
        for j in range(n):
            matrix[0][j]=0
    if first_col:
        for i in range(n):
            matrix[i][0]=0
    return matrix
```

	0	1	2	3
0	0 (0,0)	1 (0,1)	2 (0,2)	(0,3)
1	3 (1,0)	4 (1,1)	5 (1,2)	2 (1,3)
2	1 (2,0)	3 (2,1)	1 (2,2)	5 (2,3)

Rough-work

8)



(Remember case when pivot equal)

1) from R, identify pivot
2) Search for the smallest greater ele than pivot from R.
3) Swap it & reverse the arr after the pivot.

---

9) Leaders in array — O(n), O(1)

arr = [16, 17, 4, 3, 5, 2]

maxval = 17

move from right & keep updating & checking with max val.

In leaders in circular array

only max_ele is leader.

---

10) Longest consecutive Seqnn in an array

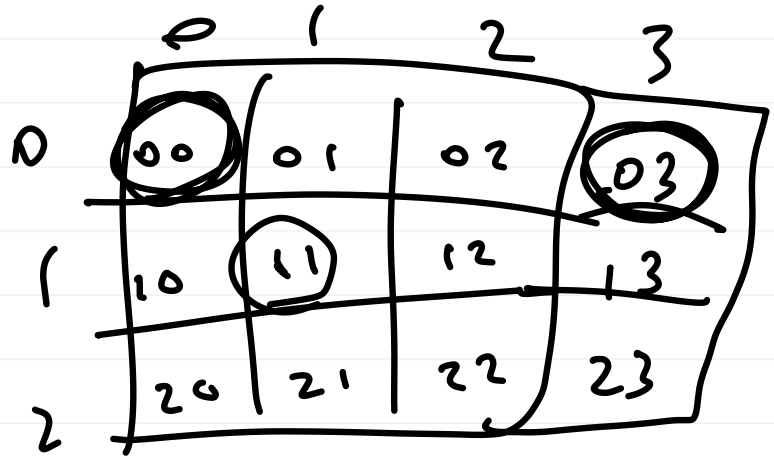
3 8 5 7 6

↳ set: {3, 8, 5, 7, 6}

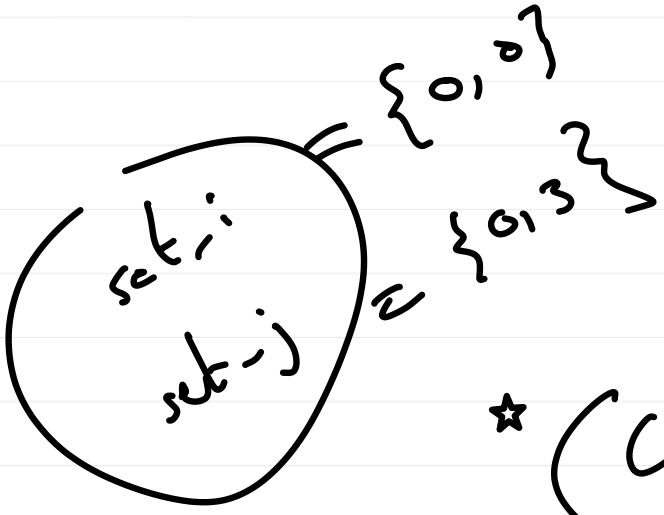
```
for num in nums:
    if num in set:
        curr = num
        while curr in set:
            curr += 1
        len = curr - num.
        if len > max_len
            start = num
            max_len = len.

return [start+i for i in range(0, maxlen)]
```

11) Set Matrix Zeroes.



→ O(k).



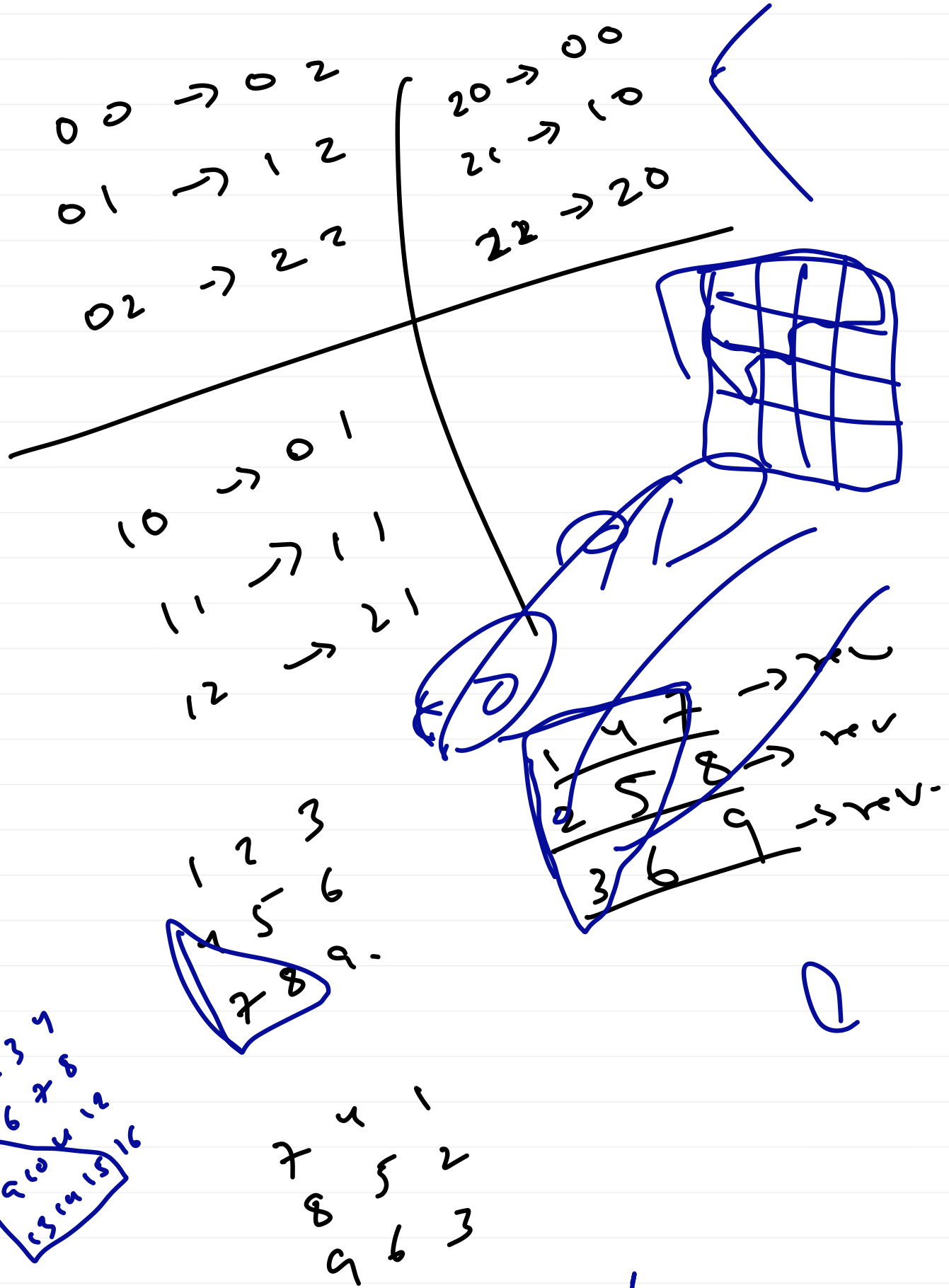
* (Check out O(1))

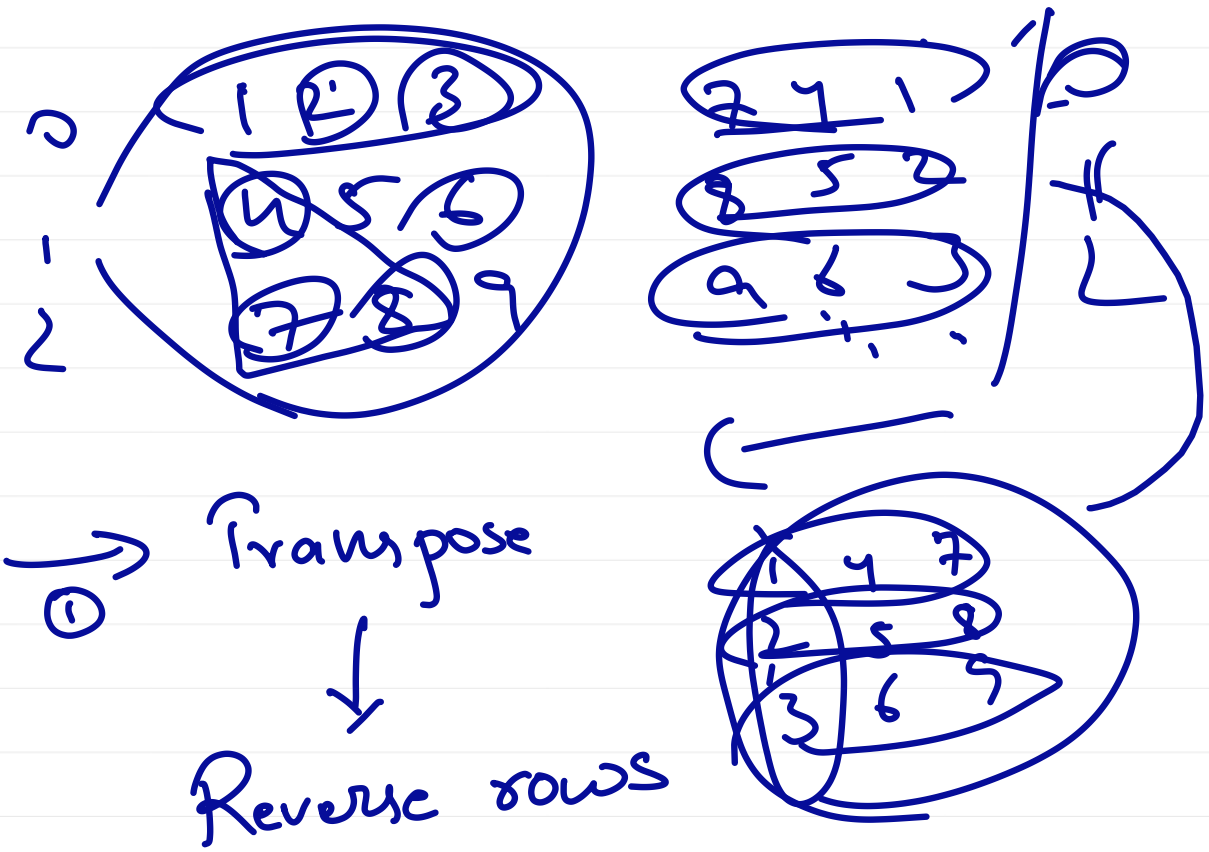
Great one

12) Rotate matrix by 90 Deg:-

00	01	02
10	11	12
20	21	22

3//2 1b/2


00 → 02
01 → 12
02 → 22
20 → 00
21 → 10
22 → 20
10 → 01
11 → 11
12 → 21
1 2 3
4 5 6
7 8 9
1 4 7
2 5 8
3 6 9
rev
rev.
rotate by 90° → transpose + rotate rows.



---

13) Spiral-matrix

```
          l   r
      t [ 1  2  3 ]
i/p :   [ 4  5  6 ]
      b [ 7  8  9 ]
```

o/p 1 2 3 6 9 8 7 4 5

Sliding Window & Two Pointers

Medium

Hard

Binary Search

Easy

Medium

Hard

Stacks & Queues

Easy

Medium

Linked Lists

Backtracking

Heap / Priority Queue

Greedy Algorithms

Dynamic Programming

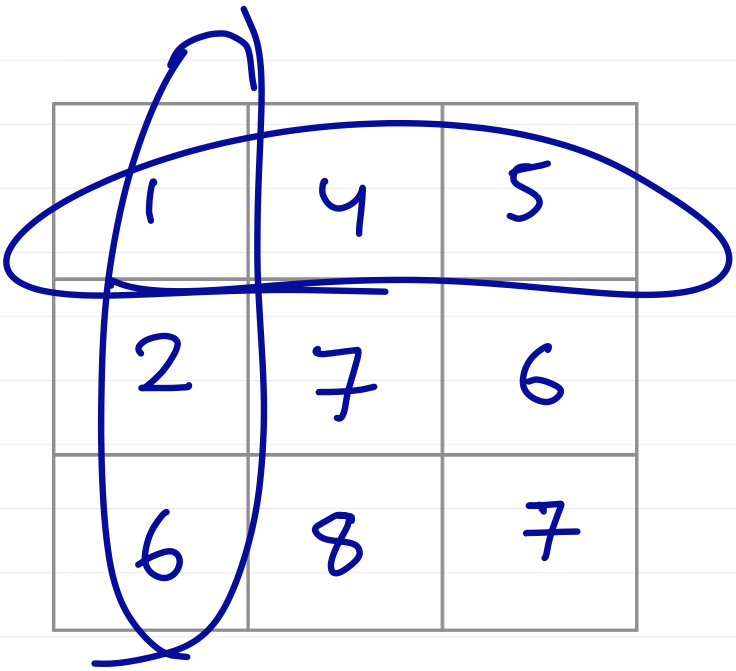
10	33	13	15
22	21	4	1
5	6	2	3
0	6	14	2

10	33	13	15
55	54	37	16
60	55	56	40
66	66	70	58

1 3 1

1 5 1

4 2 1



2

3 4

6 5 7

4 1 8 3

2

5 6

11 10

(2)

2

2 2

3 4 5 6

6 1 7 11 6

4 1 8 3

2	1	3
7	6	5
13	13	14

1

2, 7, 9, 3, 1

$2^{7^{11}}$

Graphs