132 pattern :

arr[i], arr[j], and arr[k], $i < j < k$

if order arr[i] < arr[k] < arr[j] exist, then return true;

1 2 3

in array they are in 132 order.

Bruteforce (1) array →

left min → why min ?? → To increase Range of elemts, Range]

1	2	3	8	7	5	4	6
↑	↑	↑	↑				
i	j		j				

min - arr[j]

1 3

(2) find

Complexity of

Brute force → (1) $O(n^3)$, O(1) Space

n → to find leftmin

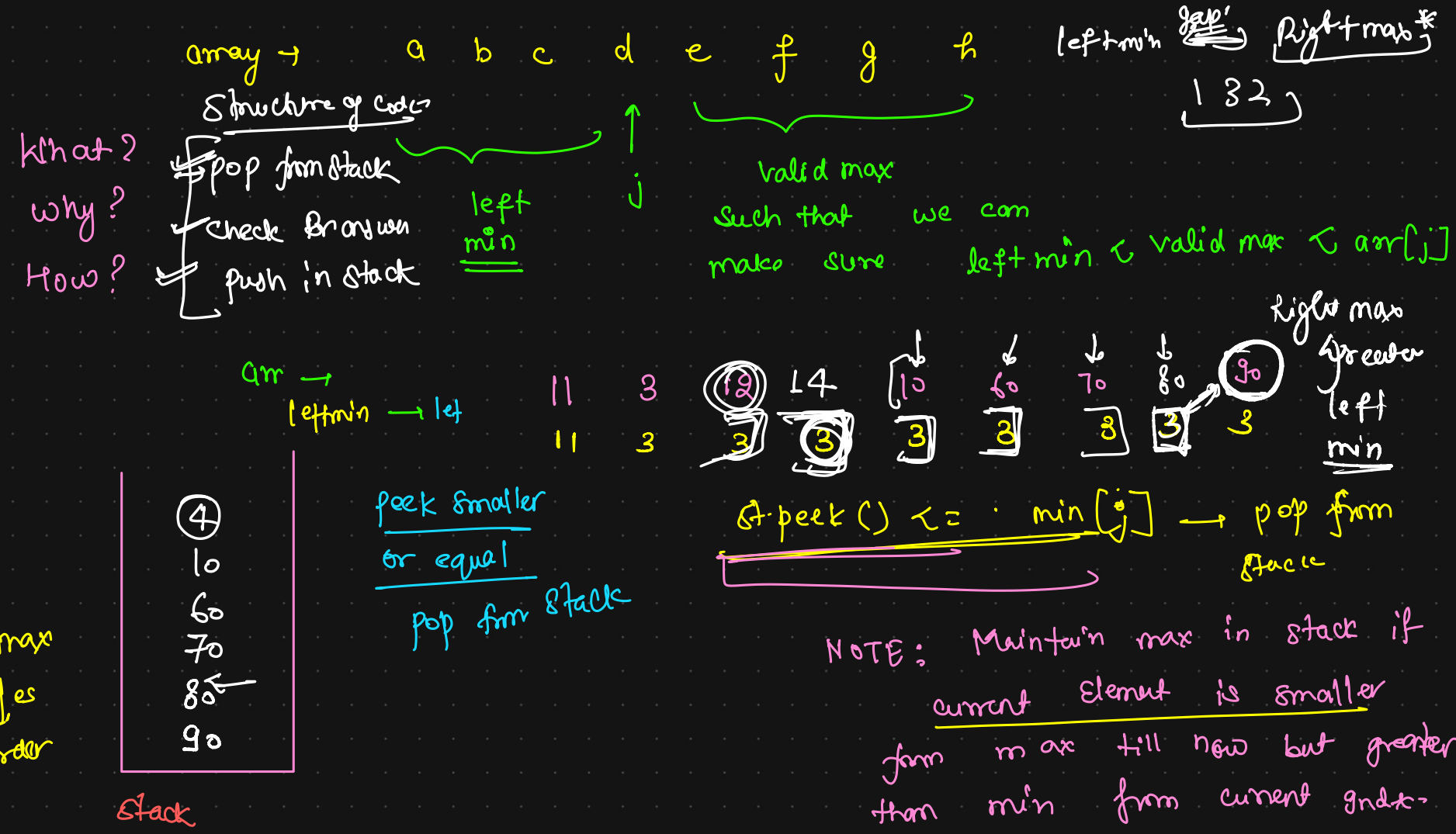
n → to find ranged elemt in Right

n → to traversal

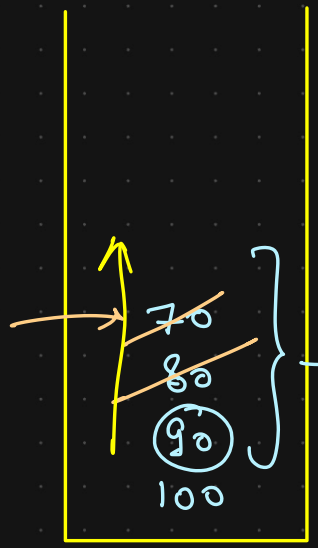
2	3	1
1	3	8
	i	j

1 32

Bruteforce 2 → Maintain left min → $O(n^2)$ - O(1) space


array → a b c d e f g h
leftmin
gap
Rightmax
1 32
Structure of code
What?
Why?
How?
pop from stack
Check Brown
push in stack
left min
j
valid max
such that we can make sure leftmin < valid max < arr[j]
arr →
leftmin → let
11 3
11 3
12 14 10 60 70 80 90
3 3 3 3 3 3 3
Right max greater left min
peek smaller or equal
pop from stack
G.peek() <= min[j] → pop from stack
max
↓es
order
4
10
60
70
80
90
Stack
NOTE: Maintain max in stack if current Element is smaller from max till now but greater than min from current index.

why?



→ Maintain Decreasing order of max such that we can select a valid and fit max for current min

pop until min is greater than max present in stack

① pop if we can't make valid Range.

② Check for result condition → True. Return

③ push in stack

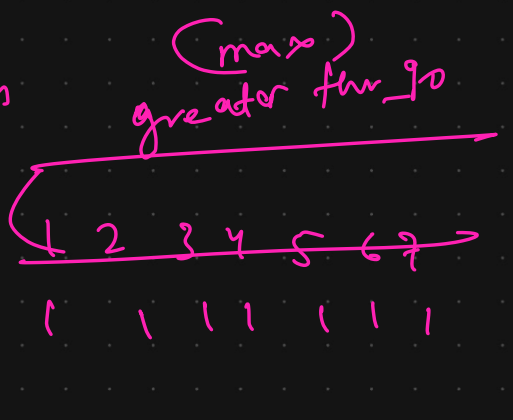
left min → array.

min Elem

90

Eg → 90 100

90



Remove k digits :

Eg → 1 2 7 4 9 1 9 6 4 2 7

increasing order

k = 3

change ← MSD → Most Significant Digit

LSD → least Significant Digit

change → massive impact

k time change

lower impact

Solve once and manage all cases.

k = ~~3~~ ~~2~~ ~~1~~ 0

1 2 1 9 6 4 2 7

1 4 3 2 9 5 6 7

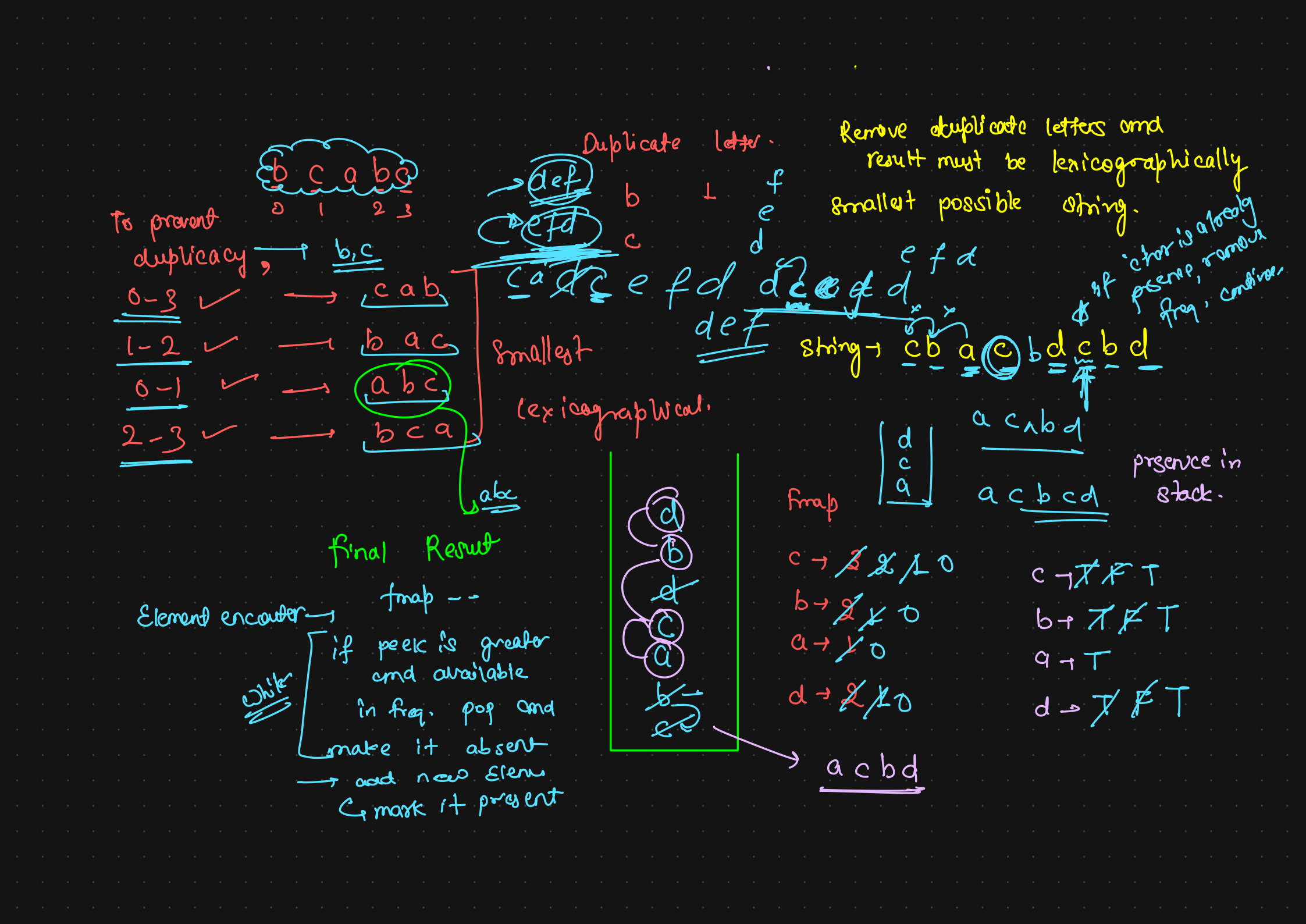
increasing

k = 4

greater

case-I → if index end and 'k' remains → remove last

case-II → if leading zero is present → avoid it


b c a b c
0 1 2 3
To prevent duplicacy, → b, c
0-3 → c a b
1-2 → b a c
0-1 → a b c
2-3 → b c a
Smallest
lexicographical.
abc
Final Result
Duplicate letter.
def
efd
cadcefd
deF
Remove duplicate letters and result must be lexicographically smallest possible string.
string → cbacbdcbd
acbd
fmap
c → 3 2 1 0
b → 2 1 0
a → 1 0
d → 2 1 0
c → T F T
b → T F T
a → T
d → T F T
Element encounter →
while
if peek is greater and available in freq. pop and make it absent → add new elem ↳ mark it present
acbd