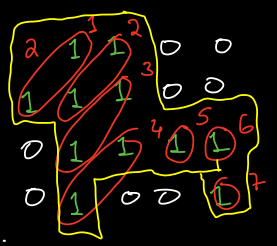
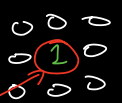
Problem: Given 2d array: 0's: empty, 1's: fresh orange, 2's: rotten orange. Every minute all fresh oranges immediately adjacent to rotten oranges will rot

How many minutes must pass until all oranges are rotten?



7 minutes

⇒ return -1 if fresh orange by itself



can never rot

↳ given several rotting oranges need to start expanding from each rotten orange

⇒ use bfs for expansion from rotting oranges

return 0 if no rotten oranges

Intro:
- Verify Constraints
- Create Testcases

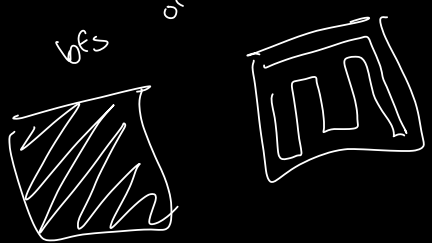
Brute Force:
- Brainstorming & Pattern Observations
- Pseudocode
- Write code
- Run through testcases
- Analyze time and space complexity

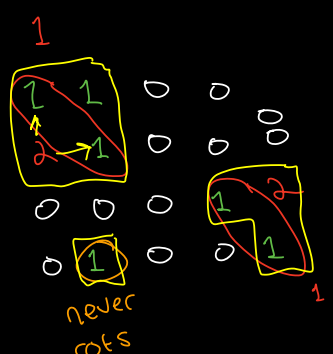
Optimal:
- Brainstorming & Pattern Observations
- Pseudocode
- Write code
- Run through testcases
- Analyze time and space complexity

sequential search to find rotting oranges

or

bfs or dfs

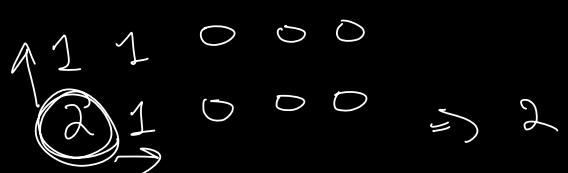


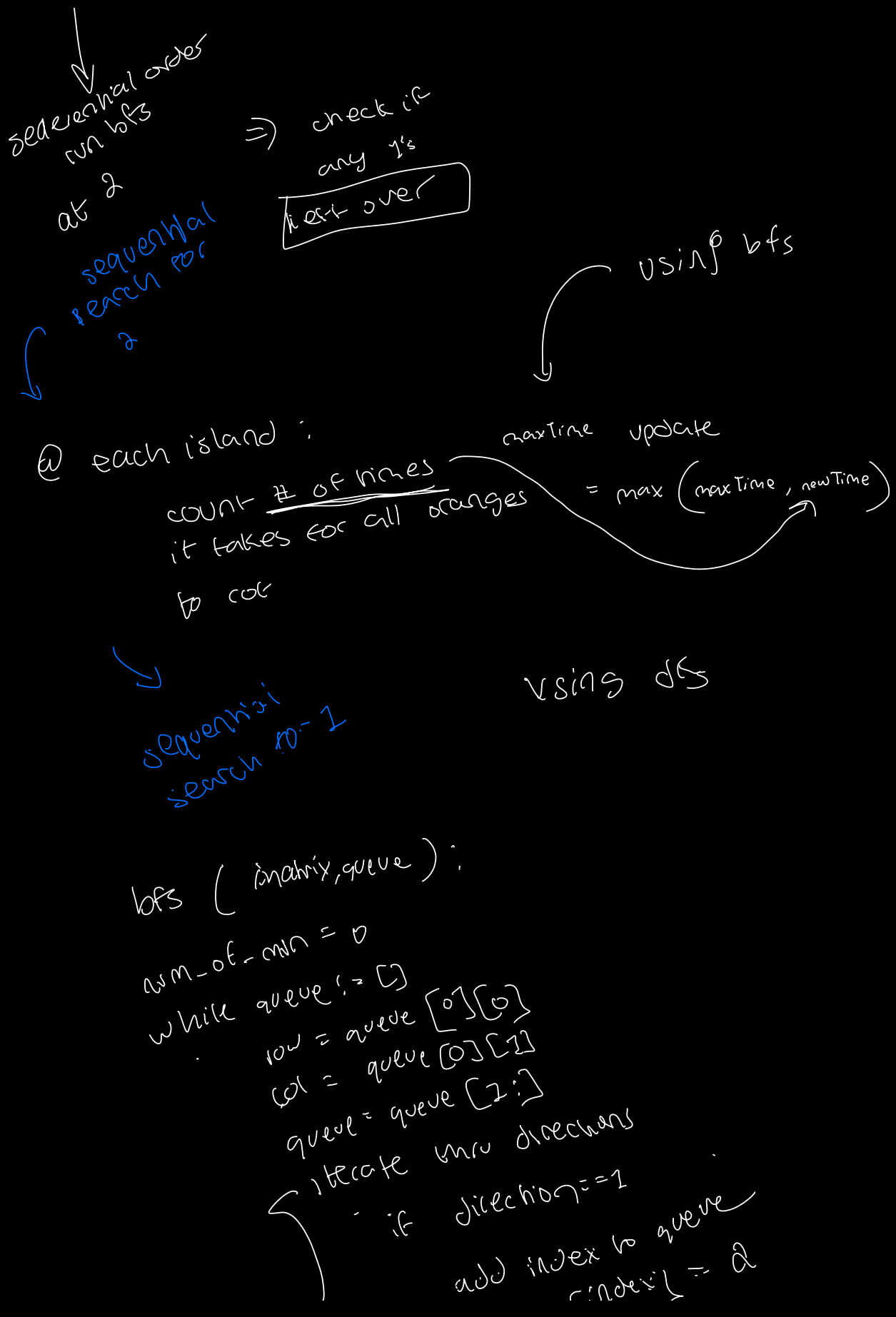


never rots

⇒ -1

run dfs each time




sequential order
run bfs
at 2
=> check if
any 1's
left over
sequential
search for
2
using bfs
@ each island:
count # of times
it takes for all oranges
to rot
maxTime update
= max (maxTime, newTime)
sequential
search for 1
using dfs
bfs (matrix, queue):
num_of_min = 0
while queue != []
row = queue [0][0]
col = queue [0][1]
queue = queue [1:]
iterate thru directions
if direction == 1
add index to queue
matrix[ ] = 2

thru all directions { max_x (...

num_of_min += 1

return num_of_min

max_min

seq search 2

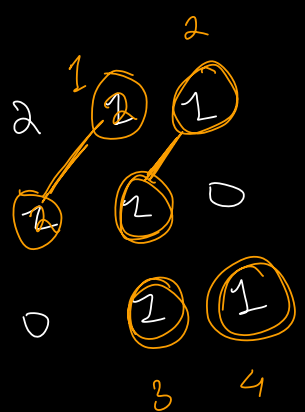
bfs

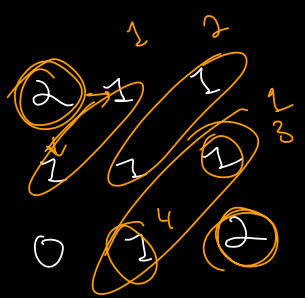
max_min = max(max_min, num_of_min)

seq search 2

return -1 if found

return max_min





find places of 2 first

find 2's

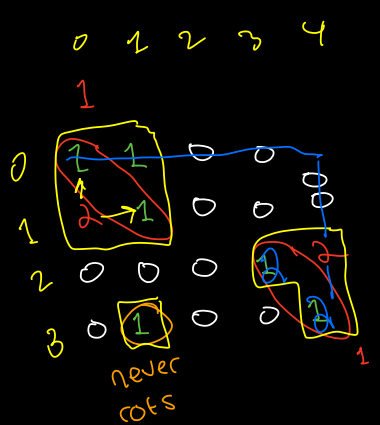
if we use dfs
how do we know
when to stop?

dfs — search - not sequential

↳ stop when dfs finished and all 2's neighbors are → 0's or 2's

run thru dfs

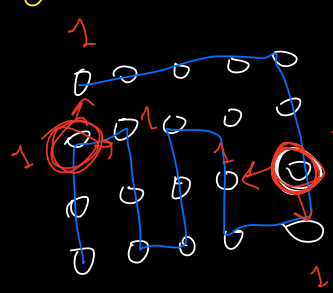
matrix or visited



[[0,1], [2,4]]

size = len(queue) = 2

count = 0



[0,1], [2,4] [0,0], [1,2], [,], [,]

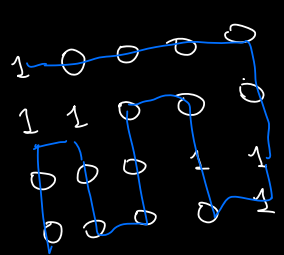
not visited and == 2 visited = True so add neighbours and mark neighbours visited = True

⇒ -1

⇒ 1st iteration

orange placement

visited



Find all initial placements of 2's

turn 2's neighbors into 2's

initialized to 0's

Sequentially iterate thru all elements

add indices of 2's to queue

seen[index] = 1

while queue not empty →

pop element off queue

go thru 2's neighbors

turn them into 2's

add indices to queue

seen[index] = 1

count += 1

sequentially search

if size == count:

num_of_min += 1

size = len(queue)

count = 0