Questions to ask me

Before Before

- Am I understanding the question correctly?
  - Have I communicated effectively and quickly with the interviewer what my understanding of the question is?

Before

Constraints?

- What is the input? What is the output?
  - Is the input sorted?
  - If we're working with numbers, are we working with only nonnegative numbers?
  - Should I return an empty list or `0` or `-1` or some boolean if the input is not valid (if the input is an empty string, list, etc.)?

Ideas and assumptions:

- ***What is the simplest, most naive approach?***
- What other approaches can I come up with?
  - Can I come up with a better approach?
  - Can I build on the naive approach?
- What are the time and space complexity of all approaches I had come up with?

During

- Is my pseudocode and explanation understood by a five-year-old?
- Can I convince the five-year-old that my idea works?
- What test cases am I going to use?
  - Did I ensure that my tests were explicitly tested on paper and adequately ran through?
  - Can I draw my solution out step-by-step using stack trace?
- Are there any edge cases?
- Am I keeping track of all the variables at each step?

After

- What were the steps that the optimal solution took from the book?
    - Did I try and explore those steps?
    - If I didn't explore those steps, what would it take for me to recognize that was the next step?
- When looking at my solution, what were the thoughts and paths I took that didn't lead to anything?
    - How can I avoid these?
    - What do I need to see first, and how can I have better intuition to see the insight?
- If I didn't get the solution, what were the insights I needed to get started?
    - Is it logical and simple enough that anyone can figure out the step I got stuck on?
- Did I try to do a magic bullet or hail mary? Can a logical train of thought prevent me from doing so?
- What can I do to get the solution faster?
    - Why did it take too long?
    - What did I spend too long on?
    - Where did it take too long?
- Was there extra thoughts that did nothing?
- Did I overemphasize this set of thoughts?
    - If it took too long, what could I have done to get the solution faster without sacrificing stability or safety?
- What did I not spend enough time on?
- Could adding more time here help prevent mistakes that I made later?
- Should I invest more time into this area to ensure I am at a lower risk of mistakes later in this step?
- Why did I write this crappy line of code?
    - Why was this comparison wrong?
    - Could I have planned this better?
    - What decisions led up to this wrong code or crappy line?
    - Could I have made a better decision earlier up in the chain?
- What edge cases do I need to look out for?
    - Could I have tested for this?
    - Was this obvious earlier on?
    - If there are too many edge cases, can I simplify my design and approach to be more systematic, so I don't have to worry about remembering so many edge cases?
- Why was my solution inadequate?
    - What went wrong? What can I do to avoid this in the future?

r = [1, 3, 1, 2] , n = 4

⇒ [1, 2, 3, 4, 1, 2]

returns [1, 2]

---

r = [3, 1, 2, 3, 1] , n = 3

⇒ [3, 1, 2, 3, 1]

↳ returns [1, 3]

Take the starting position.

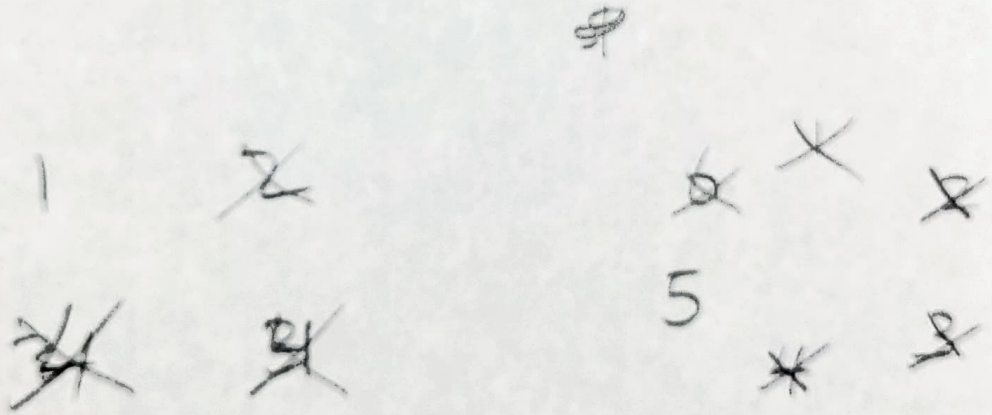
---

r = [1, 4, 2, 3], n = 4

[1, 3, 1, 2, 2, 4] Append # in between

IF the r[0] exists towards the end of the list, return the sliced, sorted list starting from the value that equals r[0] to the end of the list.

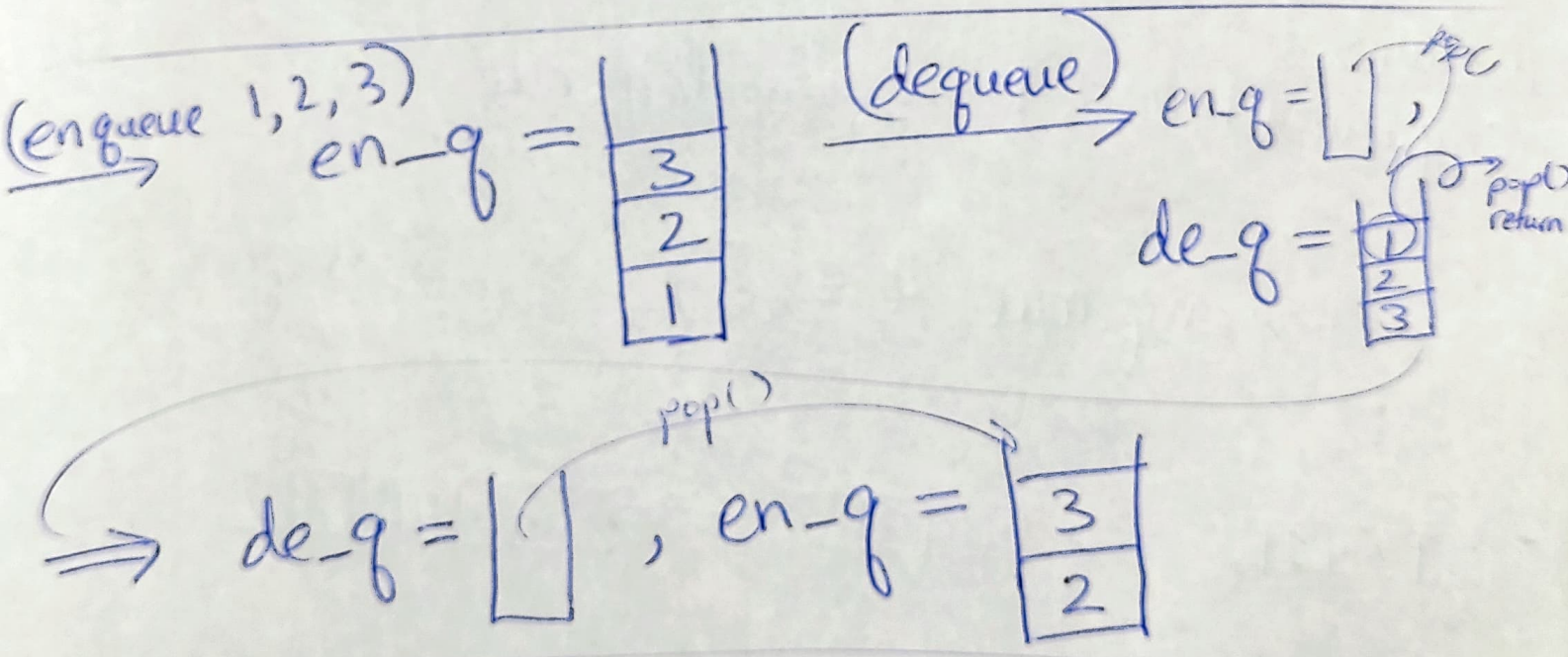
↳ slice list
  ↳ Check to see if #'s between
  ↳ return sorted.



8.8 Implement Queue using stacks API

4/10
8.8

- To enqueue, push elem. to be enqueued in one stack.
- ~~To dequeue~~, the elem. to be dequeued is then at the bottom of the stack above.
  - ↳ To dequeue, we need to pop all the elements and push them to another stack
    - ↳ Then we can pop() the elem. at top and push back the remaining elem. to the other stack.



- The problem is that dequeue ~~that~~ takes $O(n)$ time since we are push and pop'ing every elem. (while $O(1)$ time for enqueue)

1/3

→ (O(n))

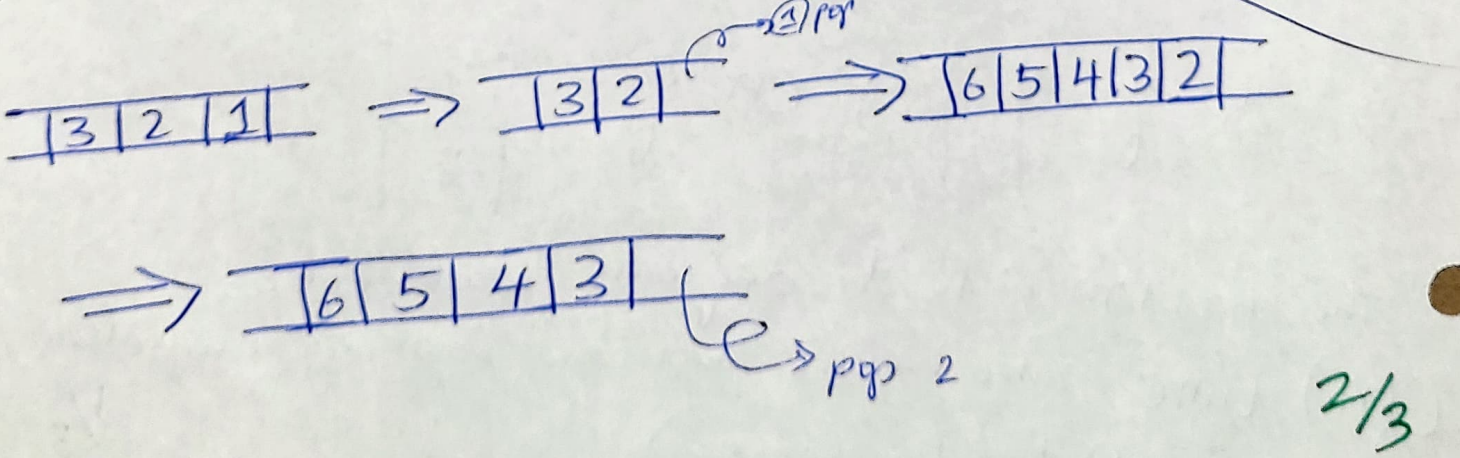
The idea to avoid such complexity is to not even push back the elem from de_q stack to en_q stack.
We can dequeue from de_q stack and enqueue from en_q stack.
Once de_q is empty, push elem from en_q to de_q.

Example operation

Queues

enqueue(1,2,3) ⟶ dequeue (⇒ 1)

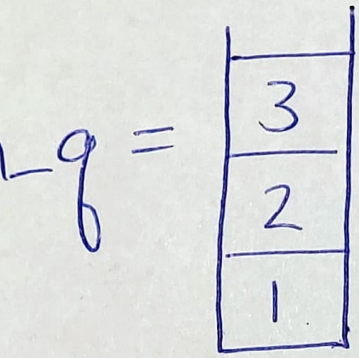
⟶ enqueue(4,5,6) ⟶ dequeue (⇒ 2)



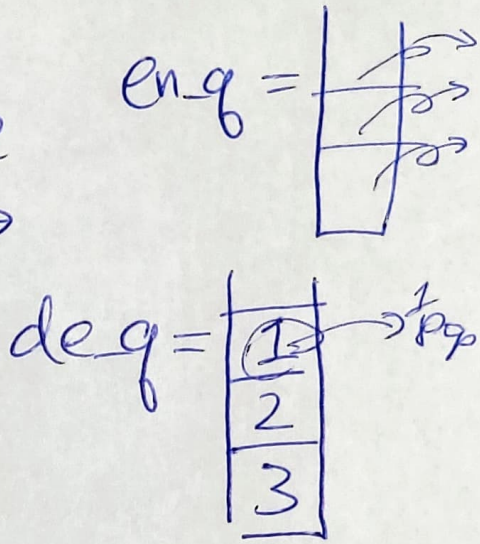
Ex. (continued)

Queues w/ stacks

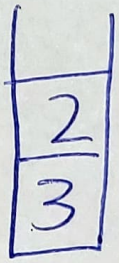
enqueue(1,2,3) →



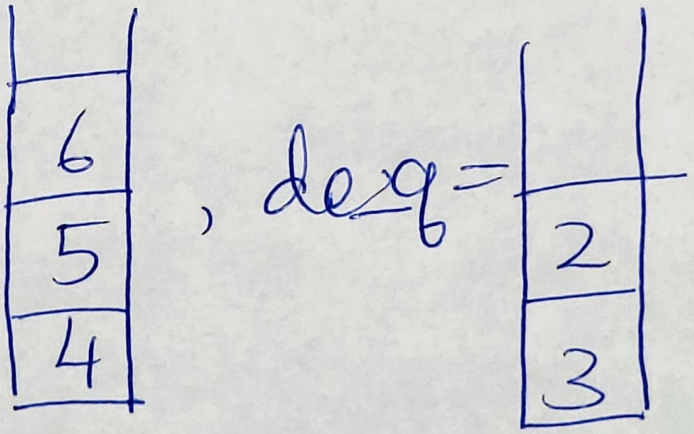
dequeue →



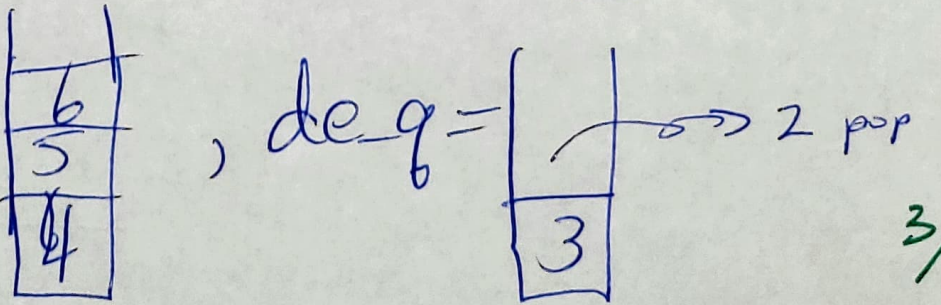
$\Rightarrow$ en_q = [ ], de_q =



enqueue(4,5,6) →

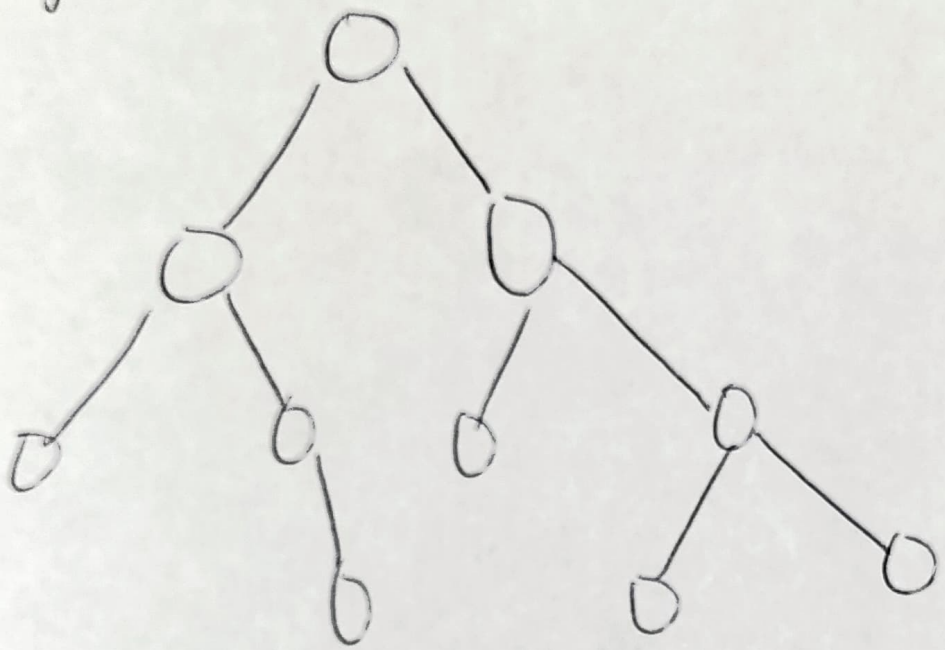


dequeue →



3/3

height-balanced bin. tree



Input: root of bin tree
Output: Boolean

Go through tree by depth and check to see if node has children?