Assignment 4- Stacks + Queues

Amalia Karaman

1) Stacking
→ Given an empty stack, what'll be the contents of stack after following operations?

1. push(8) → [8]
2. push(2) → [8, 2]
3. pop() → [8] (remove 2)
4. push(pop()×2) → [] → [16] (pop 8, ×2)
5. push(10) → [16, 10]
6. push(pop()/2) → [16, 5] (pop 10, ÷2)

Fin: [16, 5]

2) Queueing
Given empty queue, what are contents of stack after following operations?

1. push(4) → [4] (push 4)
2. push(pop()+4) → [8] (pop 4 + add 4)
3. push(8) → [8, 8] (8)
4. push(pop()/2) → [8, 4] (pop 8 + divide by 2)
5. pop() → [4] (remove 8)
6. pop() → [] (remove 4)

Fin: []
empty queue, no values remain

Discussed
3) Using a doubly linked list, you can search an element in O(n), same true for dequeue.
Given a deque q + an element y, provide an algorithm that finds the position in the deque in which element x is stored in O(n).

3) (cont.) Pseudocode:

```
int findInDeque(Deque<Integer> q, int x) {
  int index = 0
  for(int num: q) {
   if(num == x) {
     return index; // returns position
   }
   index++;
  }
  return -1; // if not found
}
```

worst-case time complexity for deque

A deque allows you to insert or delete from both ends but you still have to search through each element sequentially which resonates w/ the way worst-case scenarios operate by scanning all n elements, so I'm using O(n) time complexity. :)

I started @ index = 0, to find the position of x in the deque. Then it went through each element individually at a time, and as soon as it'd find x, it'd return the index. Otherwise, the index continues increasing as it moves through the deque, and if it never finds x, it returns -1 to indicate that it's not in the deque.

7) For algorithms written in #4-6, explain their time complexity & space complexity in Big-O notation. Explain how.

#4) Balanced Brackets
Time: O(n)
Space: O(n)

Time: Input is scanned one by one & each bracket is processed once, & each character is push/popped onto/from the stack. Pushing/popping takes O(1) time so for n brackets it's O(n).
Space: In worst case all n brackets are stored in the stack so max stack size is O(n).

#5) Decode String
Time: O(n)
Space: O(n)

Time: Since stack operations take O(1) time, this time complexity is O(1) also b/c each character processed @ MOST twice.
Space: The stack stores nested sequences & repeat counts & in worst case output is O(n) in size.

#6) Infix to Postfix
Time: O(n)
Space: O(n)

Time: It scans each ~~operation~~ character once & push/pop operations once as well, and stack operations are O(1) so we have O(n) for this algorithm.
Space: It holds the operators the stack TEMPORARILY, & each are processed once. In worst case all n operators ~~are~~ therefore could be held at once.