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

#3)

```
int findInDeque(Deque<Integer> q, int x) {
    int n = q.size();
    int leftIndex = 0, rightIndex = n - 1;
    // search from the front
    Iterator<Integer> frontIterator = q.iterator();
    while (frontIterator.hasNext()) {
        if (frontIterator.next() == x) {
            return leftIndex; // Found from the left
        }
        leftIndex++;
    }
    // search from the back
    Iterator<Integer> backIterator = q.descendingIterator();
    while (backIterator.hasNext()) {
        if (backIterator.next() == x) {
            return rightIndex; // Found from the right
        }
        rightIndex--;
    }
    return -1; // If x is not found
}
```

To find the position of x in the deque, I start by checking from both ends to minimize the number of steps. I set leftIndex to 0 (starting from the front) and rightIndex to n - 1 (starting from the back). I iterate through the deque from the left, increasing leftIndex until I find x. At the same time, I iterate from the right, decreasing rightIndex. If x is found on either side, I return the corresponding index. If x is not found, I return -1. Since I only search up to n/2 elements on average, the time complexity is O(n/2), which simplifies to O(n).

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 once.
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 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 TEMPORARILY, & each the stack are processed once. In worst case all n operators/parentheses could be held at once.