Stacks and Queues

1.

push(8)	:	8
push(2)	:	2, 8
pop()	:	8
push(pop()*2)	:	16
push(10)	:	10, 16
push(pop()/2)	:	5, 16

2.

push(4)	:	4
push(pop()+4)	:	8
push(8)	:	8, 8
push(pop()/2)	:	8, 4
pop()	:	4
pop()	:	

3.

```
node left = tail;
node right = head;
int half = size / 2;
for (int i = 0; i <= half; i++) {
    if (left.data == x) {
        return size - 1 - i;
    }
    if (right.data == x) {
        return i;
    }
    left = left.prev;
    right = right.next;
}
return -1;
```

7.

Algorithm analysis

1. Balanced Brackets
   Time: O(n), n = length of the input string

The for loop iterates from 0 through the length of the input.
Space: O(n)
If a bracket pair is never found, then n brackets will have been pushed onto the -
stack by the last iteration of the loop.

2. Decode String
Time: O(n), n = length of the input string
Within the for loop at line 8, the iterations of the while loop at line 12 are added back onto
its index; in total, n loops are done.
The other for loop at line 22 repeats for the value of each integer (not individual digits) -
in the string. But, because this would be represented with a different variable, like k, it -
becomes a coefficient to n we can ignore.
Space: O(n)
Heavily nested expressions in the original string would increase the stack's size in proportion -
to the growing size of the input.

3. Infix to Postfix
Time: O(n), n = length of the input string
The while loop nested in the for loop will pop operators that have already been pushed.
This happens once at most for each operator in the string, which at worst this gives a coefficient to n in the runtime.
Space: O(n)
If the precedence of the operations only increases as you go and there are no parentheses, then -
the stack will grow to hold all of the operations until the end.