Elyas Ata 2/28
Algorithms Stacks & Queues.

1.) push(8): the stack contains 8: [8]
push(2): now we have 2 [8,2]
pop(): we remove top element 2, becomes [8]
push(pop()*2) we remove 8 then empty stacks
then we pop 8 back then multiply by 2
[8•2] = [16]
push(10): we stack 10 to the stack; [16,10]
push(pop()/2) we get rid of 10 then stack
becomes [16], then [10] is divided by 2
so new stack [16,5]

Elyas Ata 2/28

Stacks & Queues

2.) push(4): the stack now has 4
[4]

push(pop()+4): we remove 4 from stack
then we pop 4 to the stack to add to 4. 4+4
[8]

push(8): we push 8 into the stack we
now have [8,8]

push(pop()/2): remove top element 8, then
we will 8/2 onto the stack and pop 4
back into stack: [8,4]

pop(): we will remove 4 first from stack [8]

pop(): remove 8 from stacks [ ]

Elyas Atta 2/28

Stacks & Queues

3.) We will use a front and back pointer which we will start from the front. for example initalize a canter to 0 for position.
Compare i-th element to front and back, if left pointer is <= to right pointer if it is return to current position. then check if right n - right - 1 where n is the total number of elements in deque. while i is front search which increments. by 1. right side/rear decrement by 1. Increment current position by 1 to keep track of position.
if the element isnt fand at the cross section it will return same time complexity

```
while left <= right :
    if deque[left] == x
        return pos
    if deque[right] == x
        return len(deque) - pos - 1
    left += 1
    right -= 1
    position += 1
```