Asymptotic Analysis - countPositiveElements Function

Function Overview:

The function loops through an array A of size n and counts how many elements are positive.

```
Python Function:

def countPositiveElements(A):
    count = 0          # Line 5
    for x in A:        # Line 6
        if x > 0:      # Line 7
            count += 1 # Line 8
    return count       # Line 9
```

Cost Table Recap:

Line	Operation	Cost	How many times it runs
5	count = 0	c1	1
6	for x in A	c2	n
7	if x > 0	c3	n
8	count += 1	c4	n (worst case: all positive)
9	return count	c5	1

Effort Calculation (Line-by-Line Explanation)

Step 1:

T(n) = c1 + n*c2 + n*c3 + n*c4 + c5

We sum all the individual operations:

- Line 5: 1 time -> c1

- Line 6, 7, 8: each runs n times -> n*c2 + n*c3 + n*c4

- Line 9: 1 time -> c5

Step 2:

T(n) = c1 + c5 + n*(c2 + c3 + c4)

Combine constants c1 and c5 and factor out n from the rest.

Step 3:

$T(n) = c6 + n*c7$

Rename c1 + c5 as c6, and c2 + c3 + c4 as c7 for simplicity.

Step 4:

$T(n) <= n*c6 + n*c7$

Upper bound idea: if constants are positive, $c6 + n*c7$ is less than or equal to $n*c6 + n*c7$

Step 5:

$T(n) <= n*(c6 + c7)$

Factor out n.

Final Step:

$T(n) <= n*c8$

Combine all constants into one constant c8. Since asymptotic analysis ignores constants, this simplifies the notation.

Final Form: $T(n) = O(n)$

This shows the time complexity is linear because the effort increases proportionally to n.

Why $T(2n) = 2T(n)$?

If the array size doubles, then:

$T(2n) = 2n * c8 = 2 * (n * c8) = 2 * T(n)$

So yes, linear growth confirms this.