```
# here's the recursive version: explanation follows
def len_recur(lst: list) -> int:
  """compute length of list with recursion"""
  if lst == []:      1+1
    return 0         1
  else:
    return 1 + len_recur(lst[1:])
               1     0      1
```

recurrence relation

$T_{fun}(n)$ — time; size of input; name of function

recurrence

① reduce code to operation counts

$T_{len}(0) = 3$

$T_{fun}(n) = 2 + 1 + T_{fun}(n-1) + 1$

if sublist recursion add

Solve the recurrence ②

$T_{len}(k) = 4 + T_{len}(k-1)$

$= 4 + T_{len}(k-1) + 4 + T_{len}(k-2) + \ldots$

$= (4*k) + \sum_{j=0}^{k} T_{len}(j)$

memorize key patterns ← $O(k)$

$= 4k + O(k)$

$= O(k)$ linear

③ characterize it

```
def len_for(lst: list) -> int:
  """compute length of list with for loop"""
  count = 0      1
  for elt in lst:    2 for is-empty?
    count = count + 1
  return count     1     1
```

- if is implicit, so easy to forget in recurrence
- repeated call is implicit in for, so have to build that in by hand

is-empty; count = count + 1

$T_{for}(k) = 1 + (2+3) + (2+3) + (2+3) \ldots$

k times through loop

$= O(k) * 5$

$= O(k)$ linear

So far:
worst-case analysis