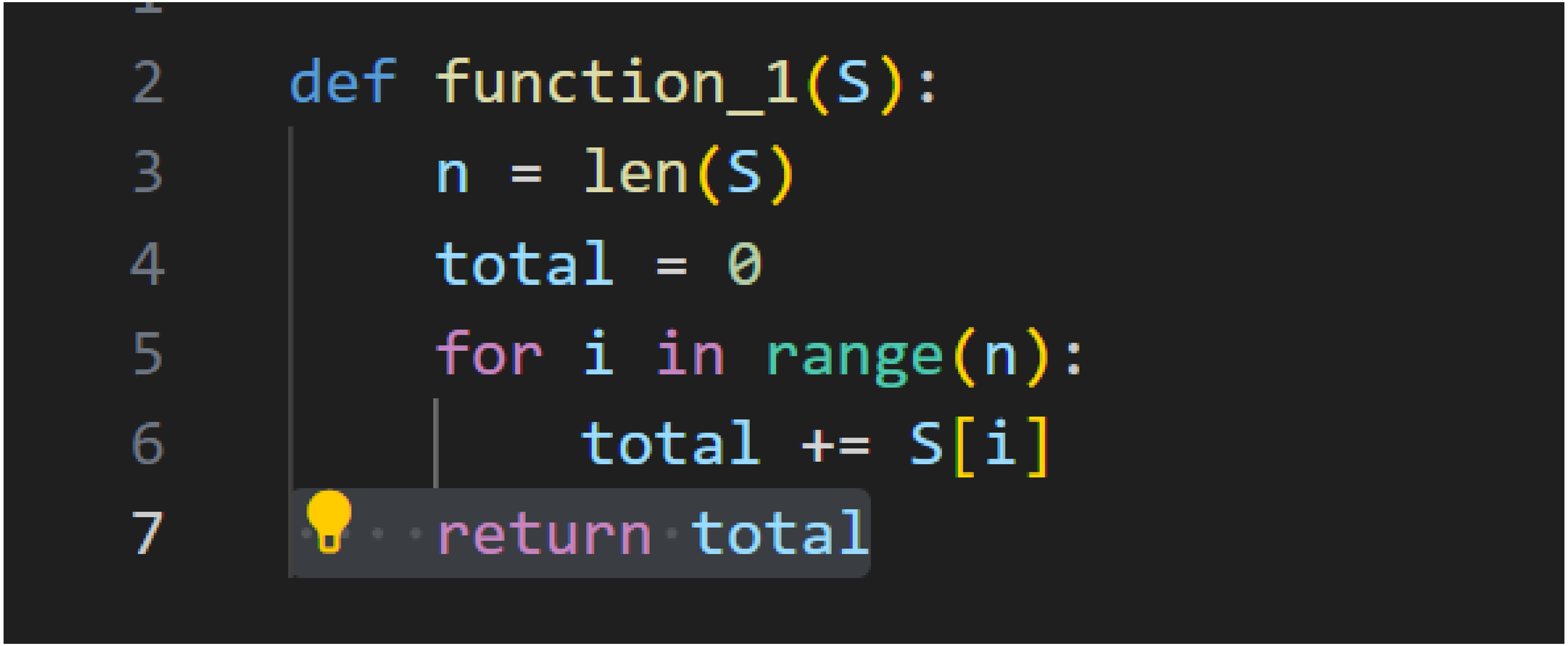
DSA3 Lab session 2

What the hell is this Big O thing?

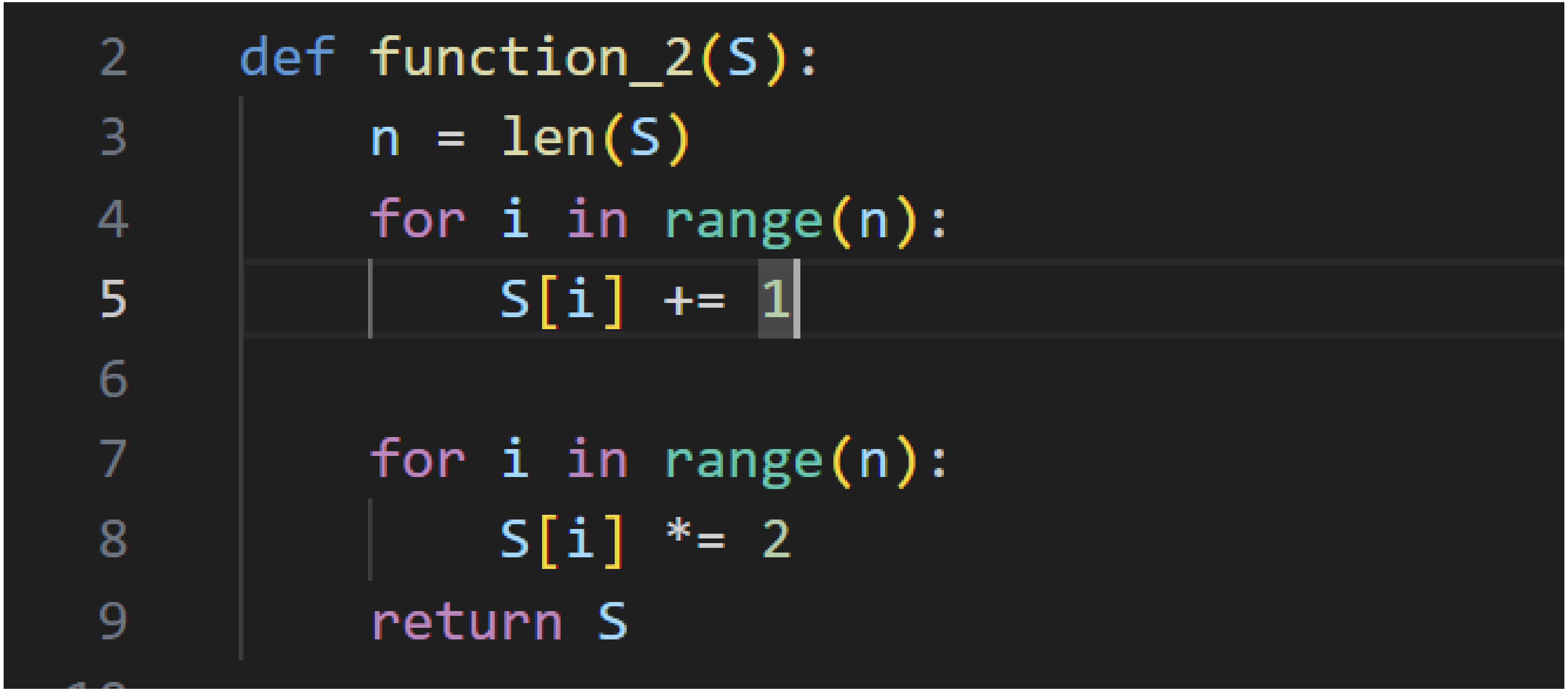
- Strong intuitive feel for Big O is critical for all software engineers
- Precise/rigorous/formal Big O important in:
  - Algorithm research
  - Competitive programming
  - Performance-critical applications
- People typically care less about Big Ω and Big Θ

Can you “feel” what the Big O is here?



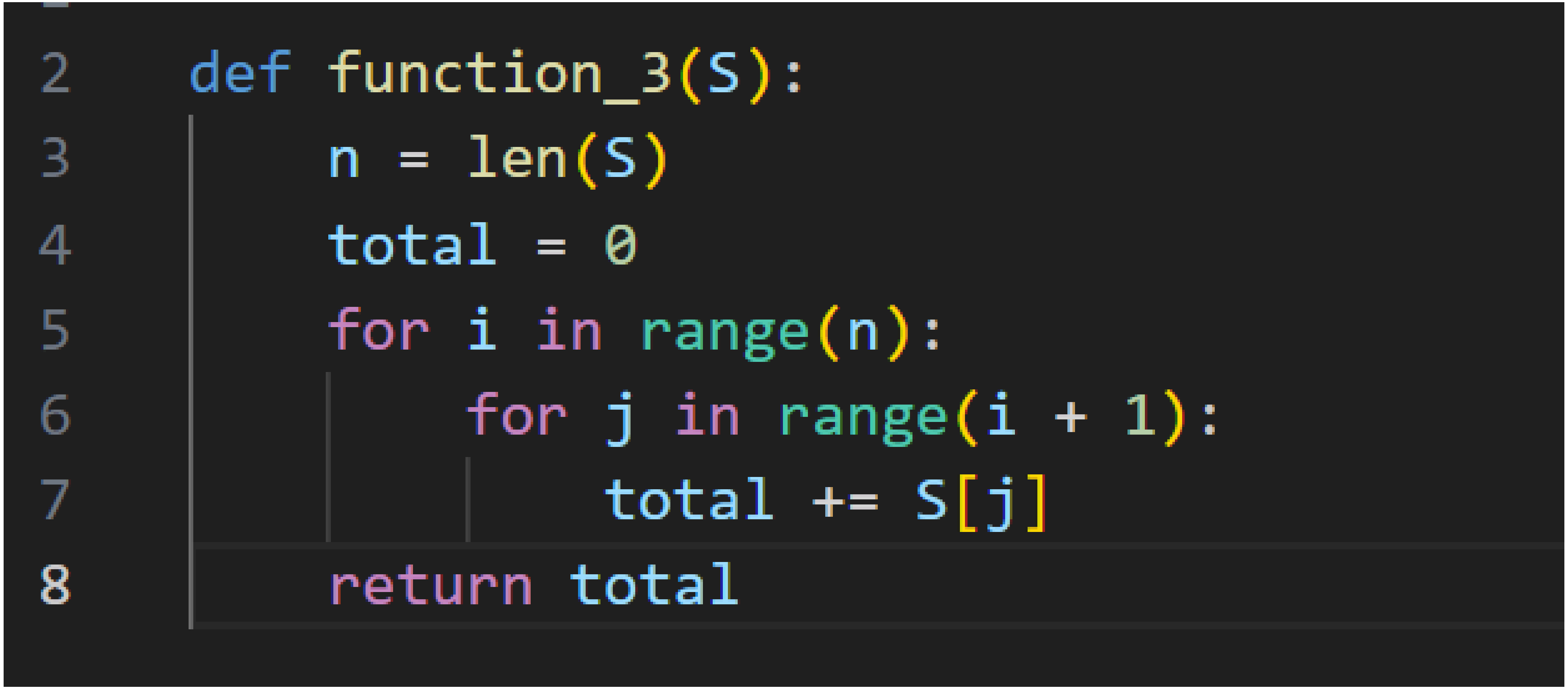
```python
def function_1(S):
    n = len(S)
    total = 0
    for i in range(n):
        total += S[i]
    return total
```

and here?



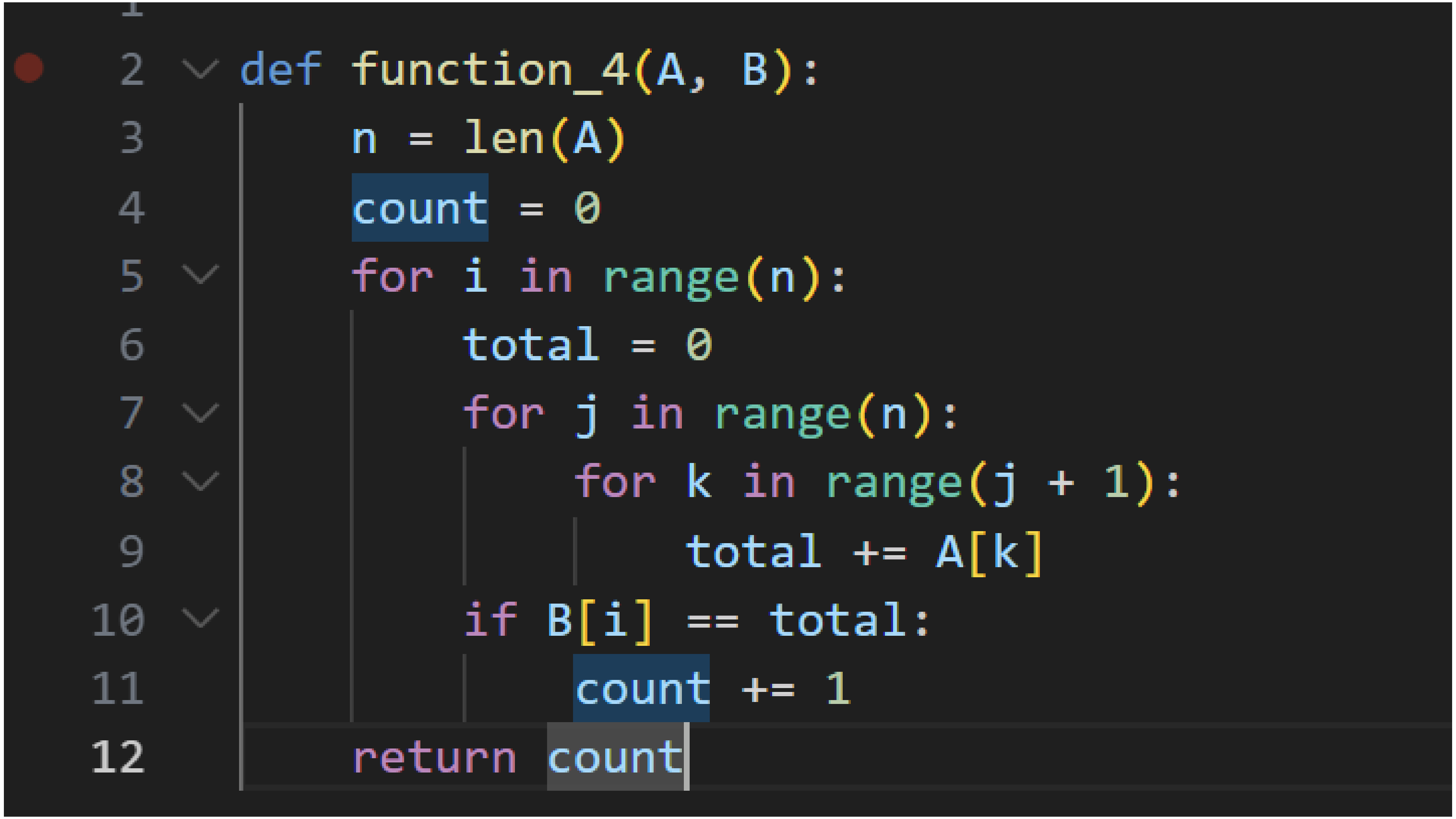
```python
def function_2(S):
    n = len(S)
    for i in range(n):
        S[i] += 1

    for i in range(n):
        S[i] *= 2
    return S
```

How about here?



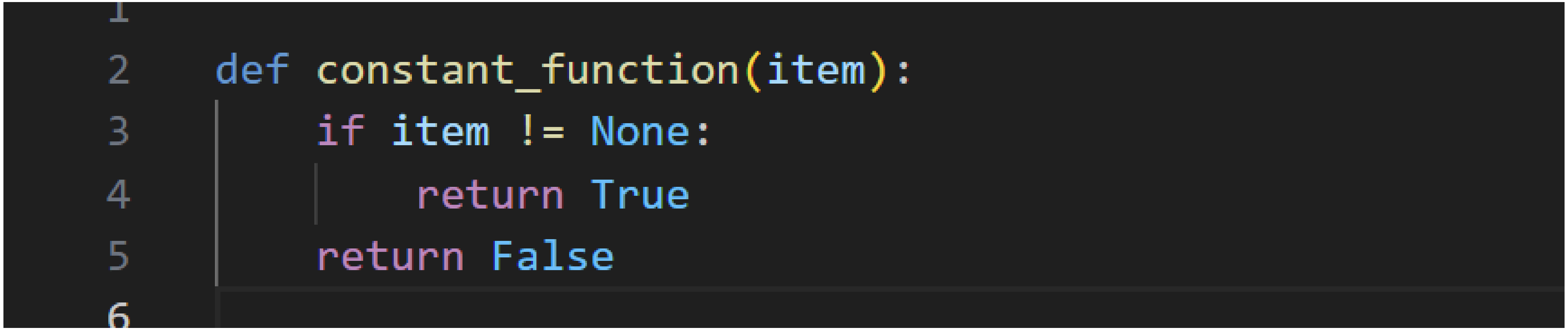
```python
def function_3(S):
    n = len(S)
    total = 0
    for i in range(n):
        for j in range(i + 1):
            total += S[j]
    return total
```

One more...



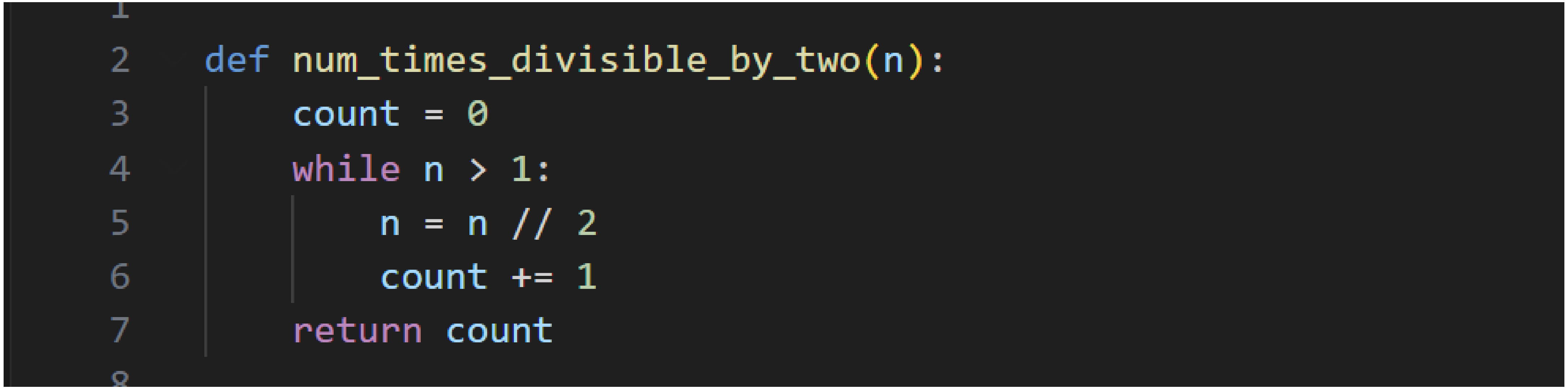
```python

def function_4(A, B):
    n = len(A)
    count = 0
    for i in range(n):
        total = 0
        for j in range(n):
            for k in range(j + 1):
                total += A[k]
        if B[i] == total:
            count += 1
    return count
```

Other common patterns: O(1)



```python
def constant_function(item):
    if item != None:
        return True
    return False
```

Other common patterns: O(log n)



```python
def num_times_divisible_by_two(n):
    count = 0
    while n > 1:
        n = n // 2
        count += 1
    return count
```

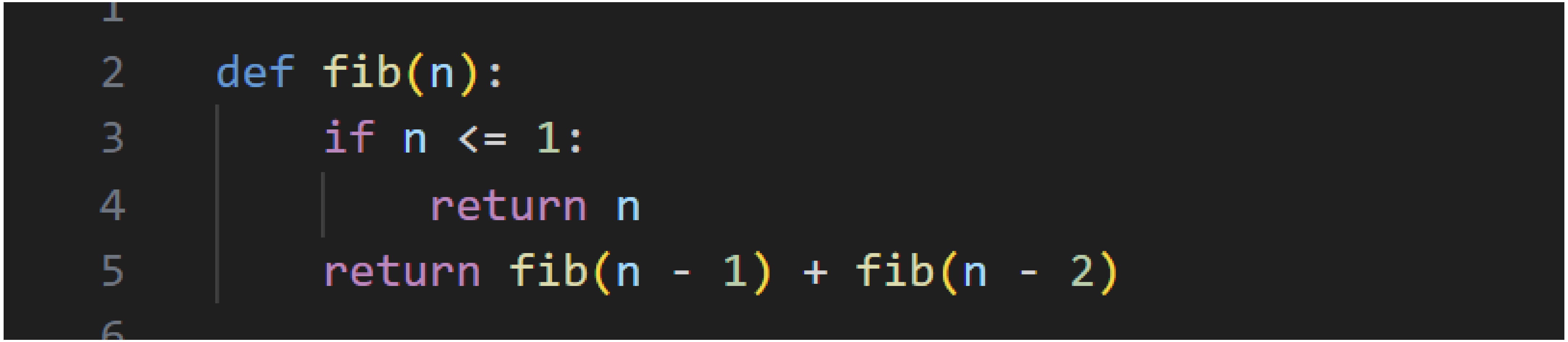
We now know O(n) and O(log n)

- How do we conceptualize

- O(n * log n)?

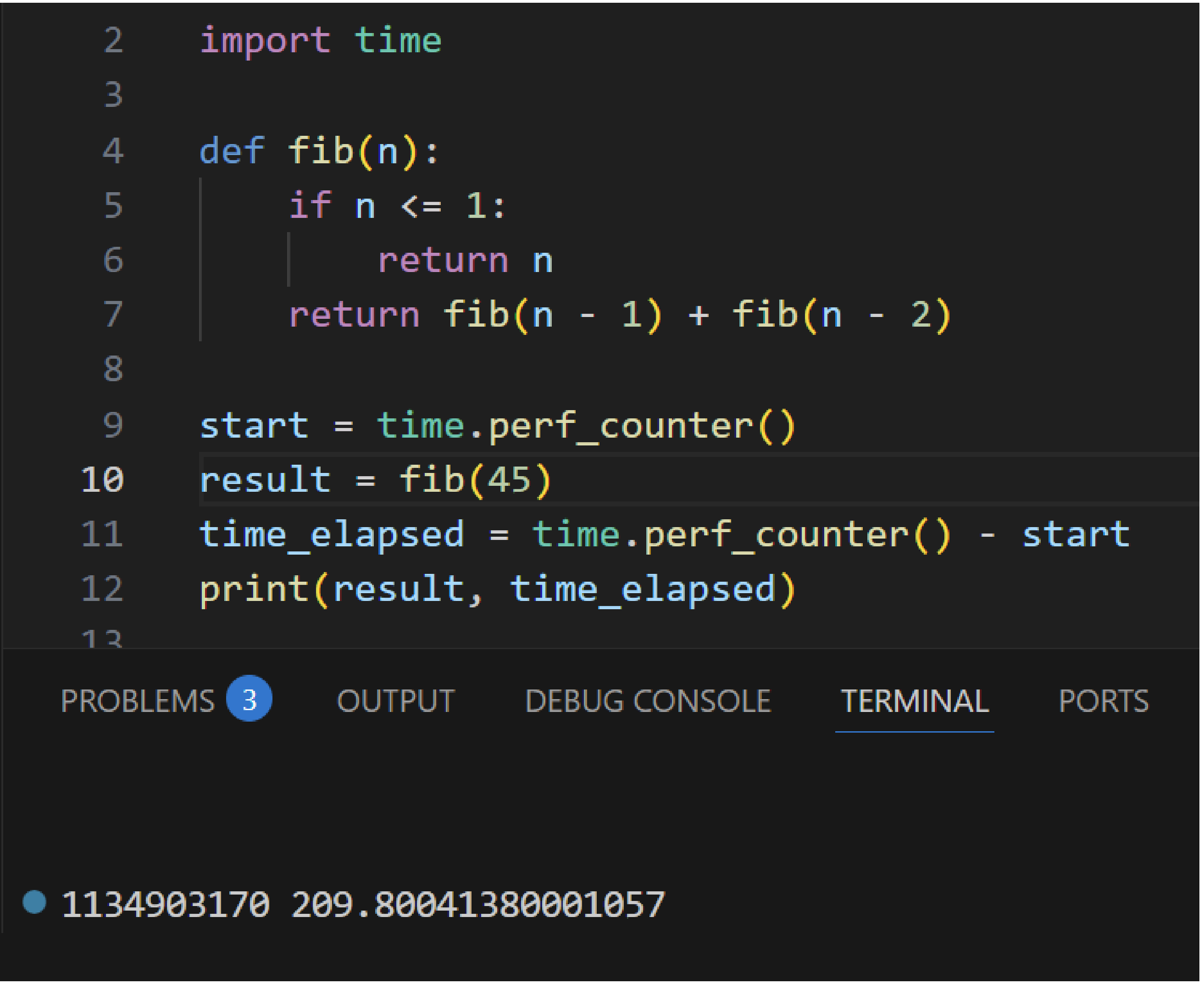
- We’ll see some O(n log n) next week

Other common patterns: $O(2^n)$



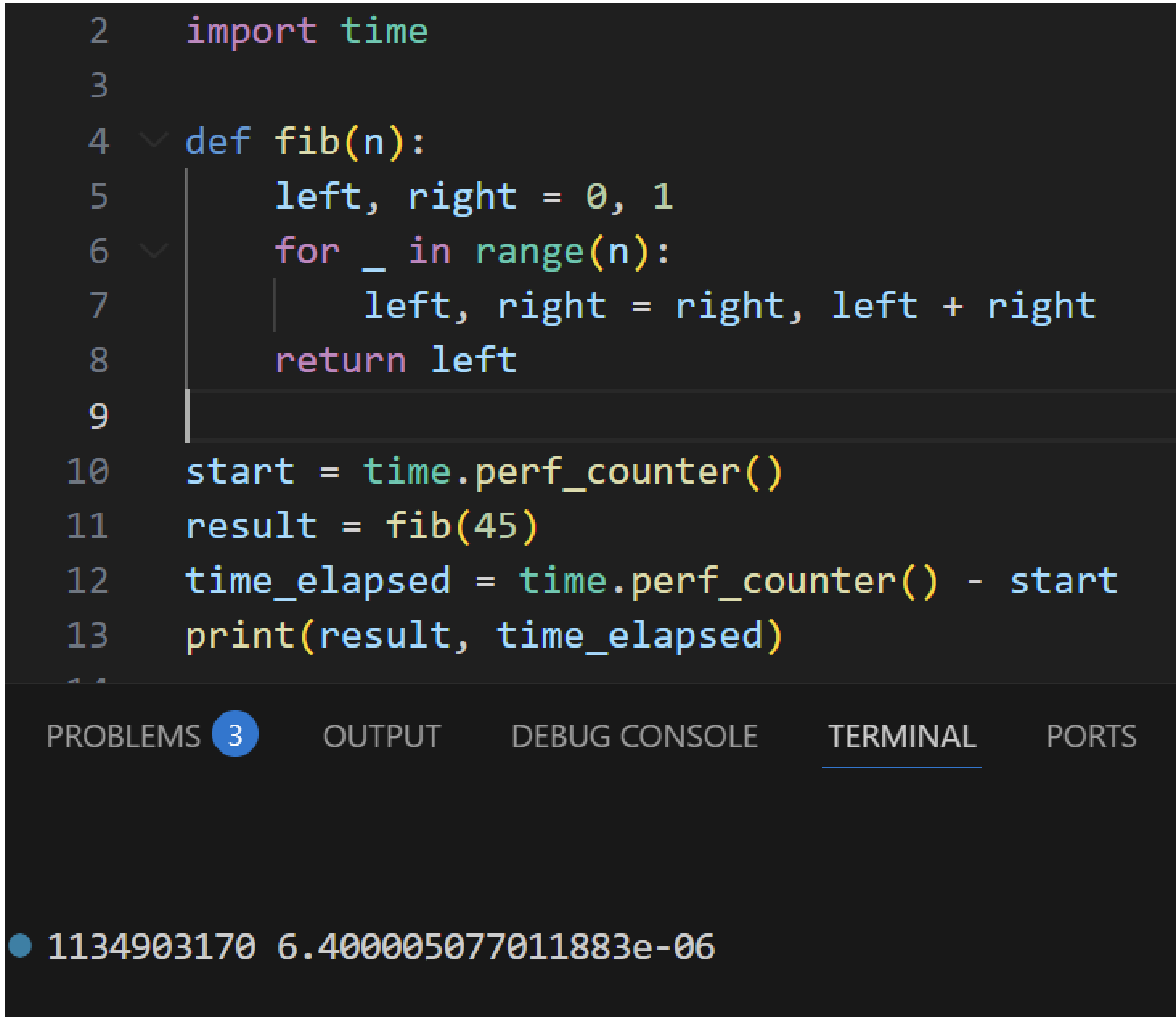
```python
def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)
```

Let’s avoid $O(2^n)$ if we can



```python
import time

def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)

start = time.perf_counter()
result = fib(45)
time_elapsed = time.perf_counter() - start
print(result, time_elapsed)
```

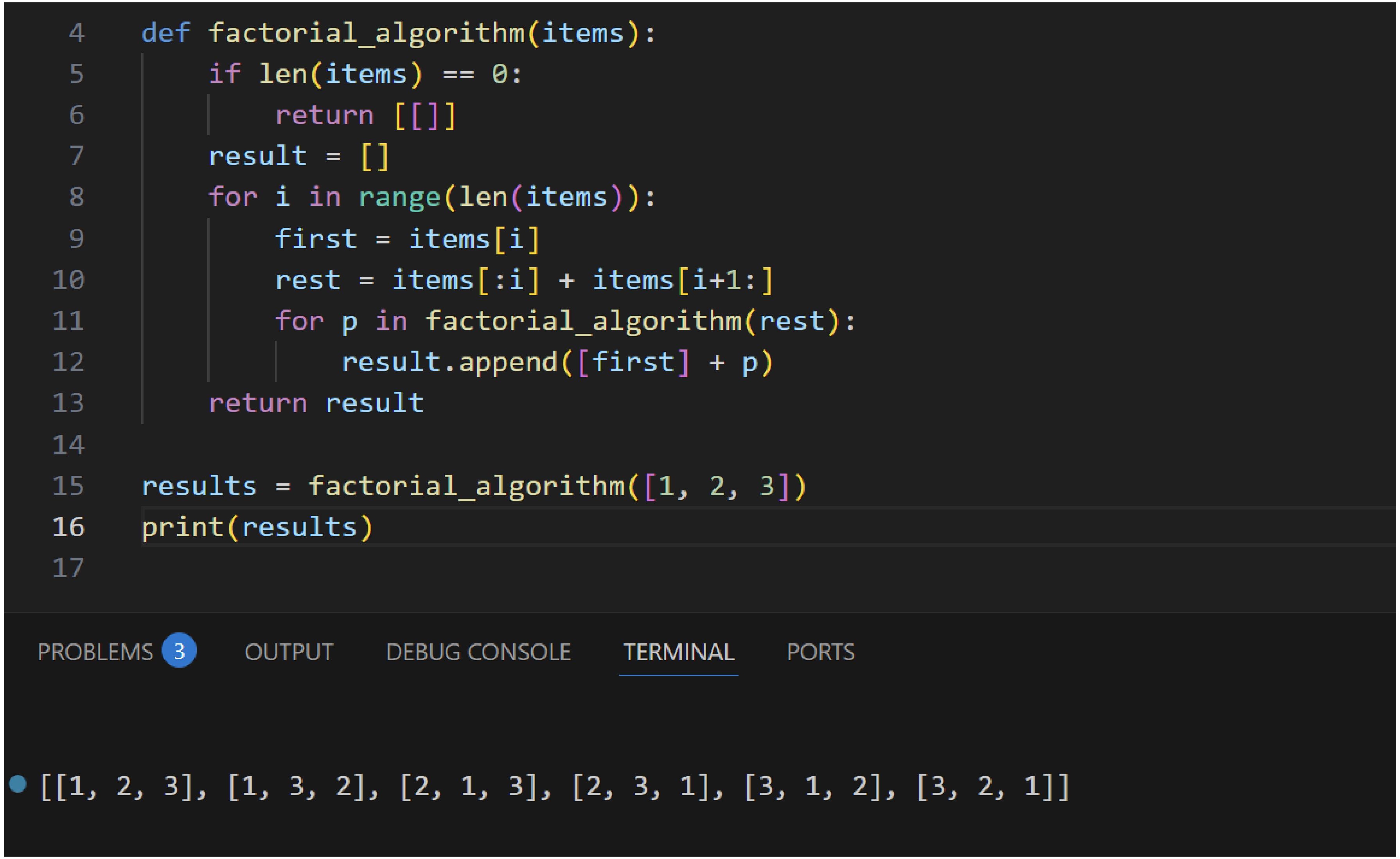
```
1134903170 209.80041380001057
```



```python
import time

def fib(n):
    left, right = 0, 1
    for _ in range(n):
        left, right = right, left + right
    return left

start = time.perf_counter()
result = fib(45)
time_elapsed = time.perf_counter() - start
print(result, time_elapsed)
```

```
1134903170 6.400005077011883e-06
```

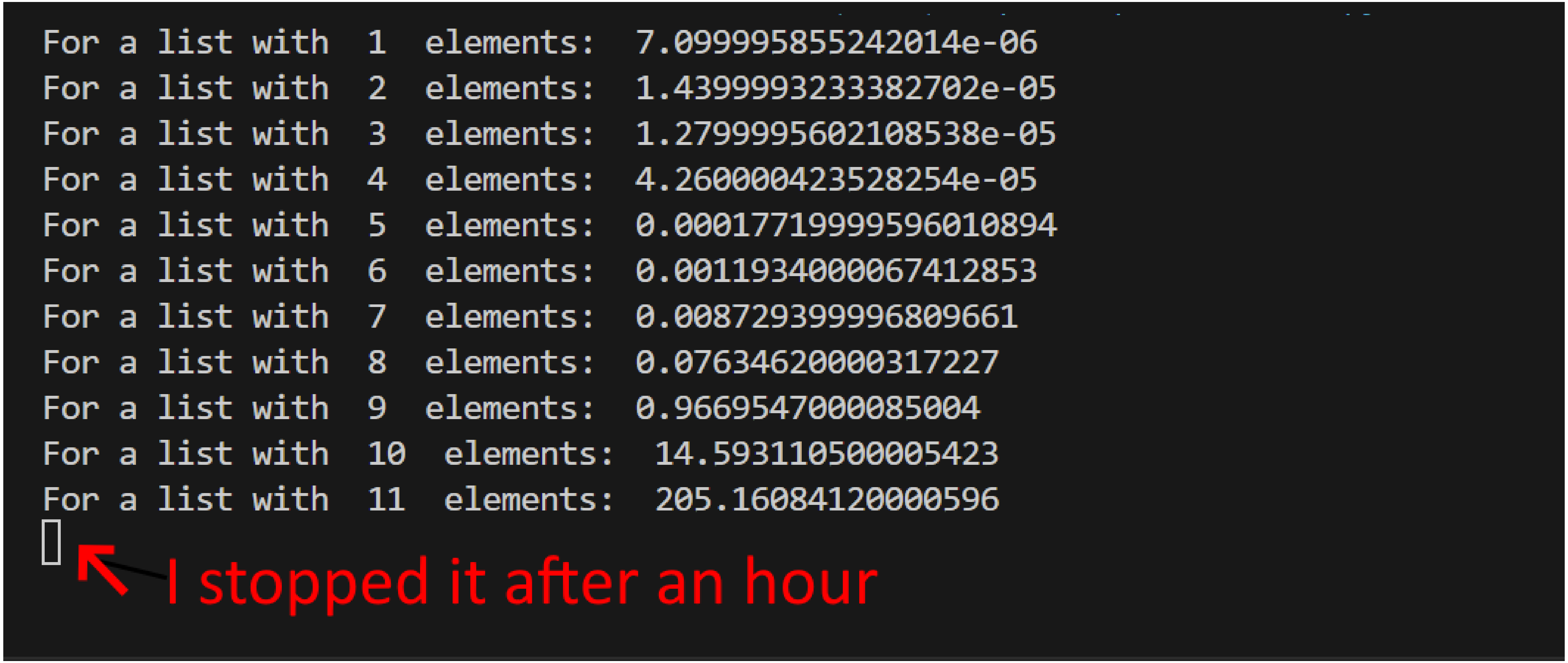
Other common patterns: O(n!)



```python
def factorial_algorithm(items):
    if len(items) == 0:
        return [[]]
    result = []
    for i in range(len(items)):
        first = items[i]
        rest = items[:i] + items[i+1:]
        for p in factorial_algorithm(rest):
            result.append([first] + p)
    return result

results = factorial_algorithm([1, 2, 3])
print(results)
```

```
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

O(n!) is bad! We need to avoid it!



```
For a list with  1  elements:  7.099995855242014e-06
For a list with  2  elements:  1.4399993233382702e-05
For a list with  3  elements:  1.2799995602108538e-05
For a list with  4  elements:  4.260000423528254e-05
For a list with  5  elements:  0.00017719999596010894
For a list with  6  elements:  0.0011934000067412853
For a list with  7  elements:  0.008729399996809661
For a list with  8  elements:  0.0763462000317227
For a list with  9  elements:  0.9669547000085004
For a list with  10  elements:  14.593110500005423
For a list with  11  elements:  205.16084120000596
▯
```

I stopped it after an hour

Traveling salesman problem

- You have a list of cities, and a list of distances between each city pair

- What’s the shortest route that visits every city once and returns to the origin city?

- This will come up later; you don’t need to study it on your own now

Is Big O our lord and master?

- Al and Bob are arguing about which of their algms is faster

- Al’s is O(nlogn)

- Bob’s is O(n^2)

Is Big O our lord and master?

- Al and Bob are arguing about which of their algms is faster
    - Al's is O(nlogn)
    - Bob's is O(n^2)

- They time their algorithms, if n < 100 O(n^2) is faster, n >= 100 O(nlogn) faster

- What's happening?

What are some use cases of recursion?

What are some use cases of recursion?

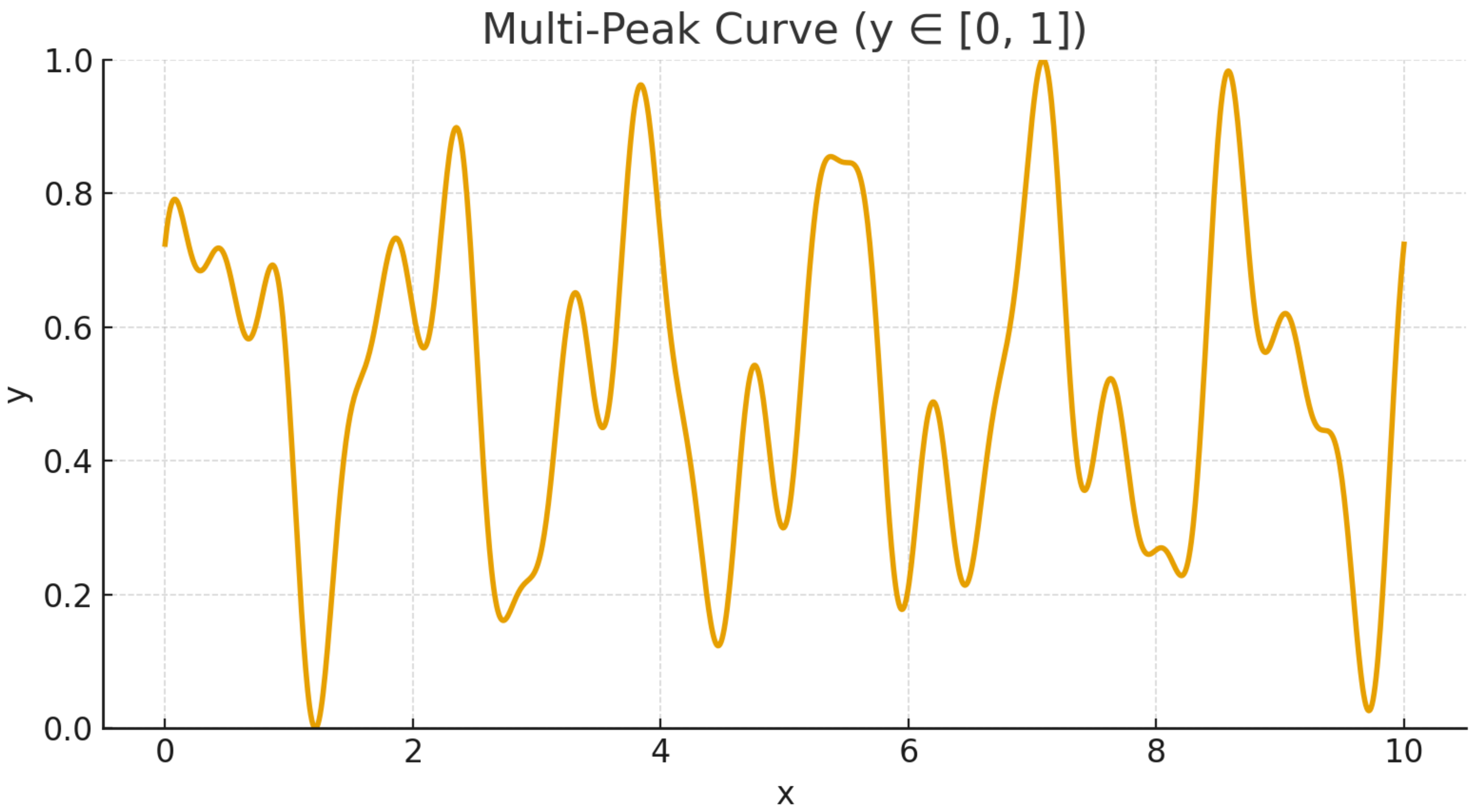
- Fractal generation
- Math problems (Fibonacci, factorial, etc)
- Tree/graph traversals
- Divide/conquer algorithms

What are some of the weaknesses of recursion?

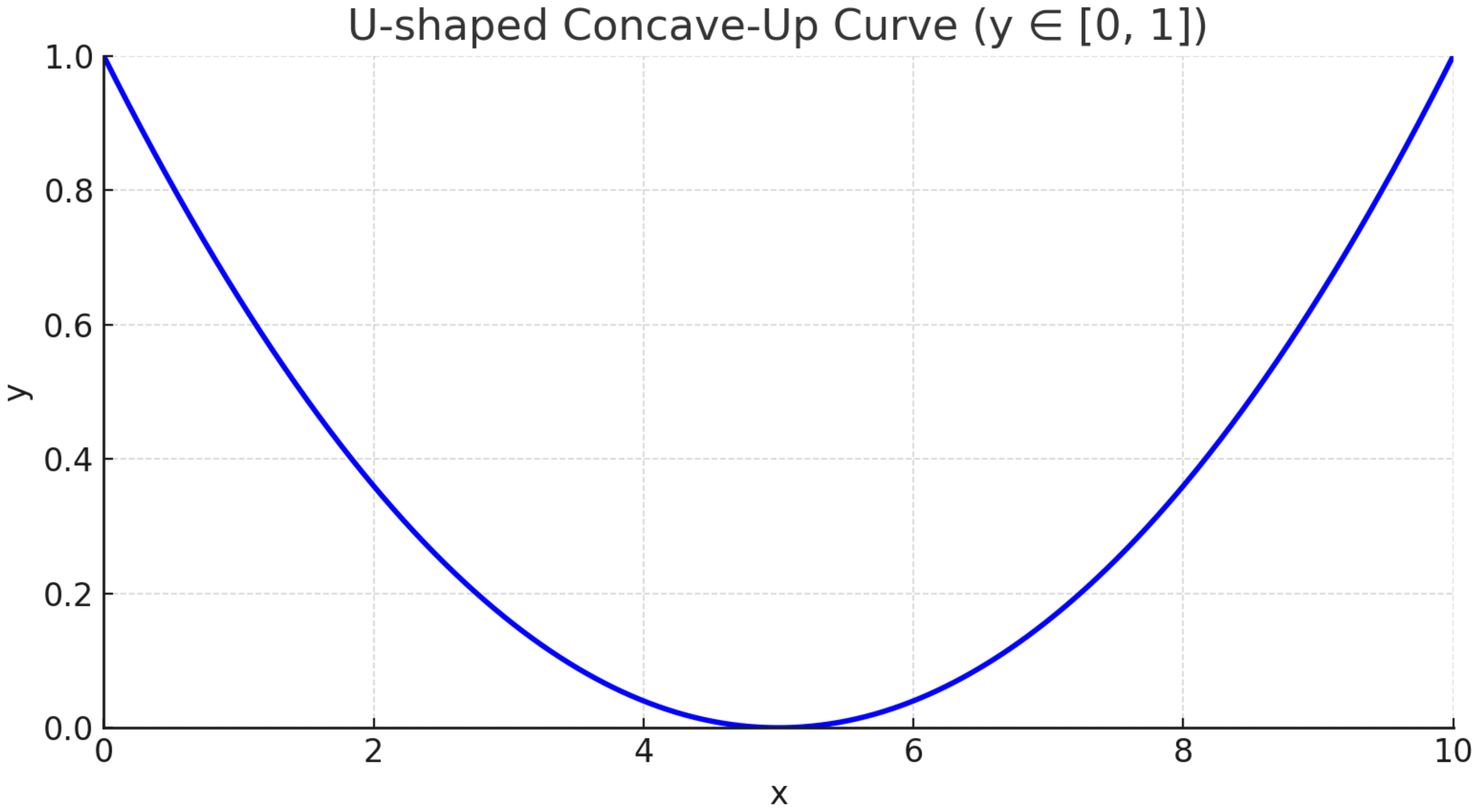
A language-independent question

- We have some list of n numbers
- We want to remove all the duplicates
- How?

Let’s take a look at assignment 2



Let’s take a look at assignment 2



ChatGPT did this, don’t @me