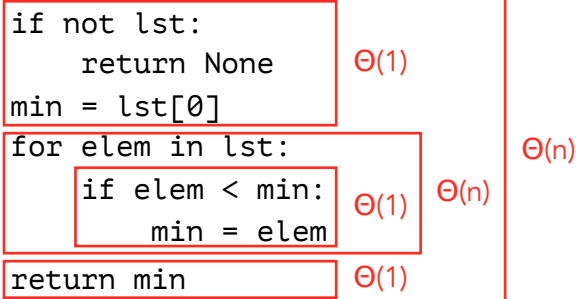


## Orders of Growth

For each function below:

- Break it up into blocks, as we did in the chapter.
- Identify the order of growth for each block, or draw the recursive tree if applicable.
- Identify the order of growth for the whole function.

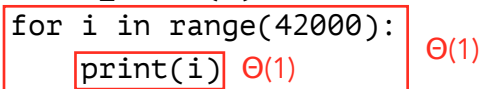
```
def min(lst):
```



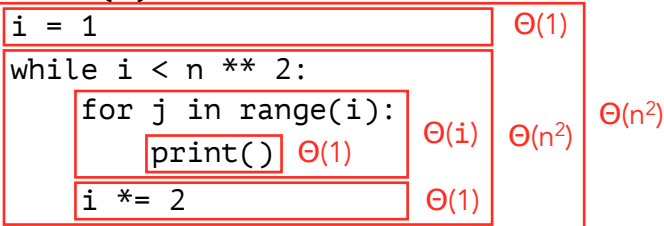
```
def captains_log(n):
```



```
def turbo_count(n):
```

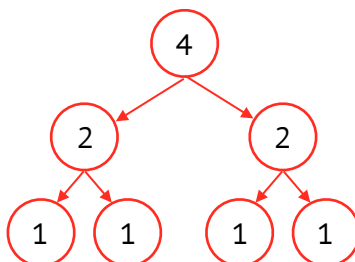
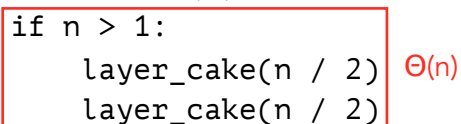


```
def blink(n):
```



This function represents the sum  $1 + 2 + 4 + 8 + \dots + n^2$ . More formally, it's the sum over  $i=[0 \text{ to } \log n^2]$  of  $2^i$ , which equals  $n^2$ . Note: This is one of the hardest analysis problems you'll see in 61A. It is worth knowing this mathematical identity, in particular.

```
def layer_cake(n):
```



In each call,  $n$  is cut in half so each node is  $\Theta(\log n)$ . However, there are two branches for each call, so there are  $2^n$  nodes. The runtime is  $\Theta(2^{\log n}) = \Theta(n)$ .

```
def fib_iterative(n):
```

curr, next = 0, 1	$\Theta(1)$	
while n > 0:		
curr, next = next, curr + next	$\Theta(1)$	$\Theta(n)$
n -= 1	$\Theta(1)$	
return curr	$\Theta(1)$	

$\Theta(n)$

```
def fib_recursive(n):
```

```
    if n <= 1:
        return n
    return fib_recursive(n-1) + fib_recursive(n-2)
```

$\Theta(\phi^n)$

$\Theta(2^n)$  is also fine here, but technically it is not as exact as  $\Theta(\phi^n)$ , where  $\phi$  is the golden ratio.

## Code Writing

Write a program called `factors` that takes in a number  $n$ , and returns a set containing all the factors of  $n$ . It should run in  $\Theta(\sqrt{n})$  at most.

```
from math import sqrt
def factors(n):
    factors, i = set(), 1
    while i <= sqrt(n):
        if n % i == 0:
            factors.update([i] if i * i == n else [i, n // i])
        i += 1
    return factors
```