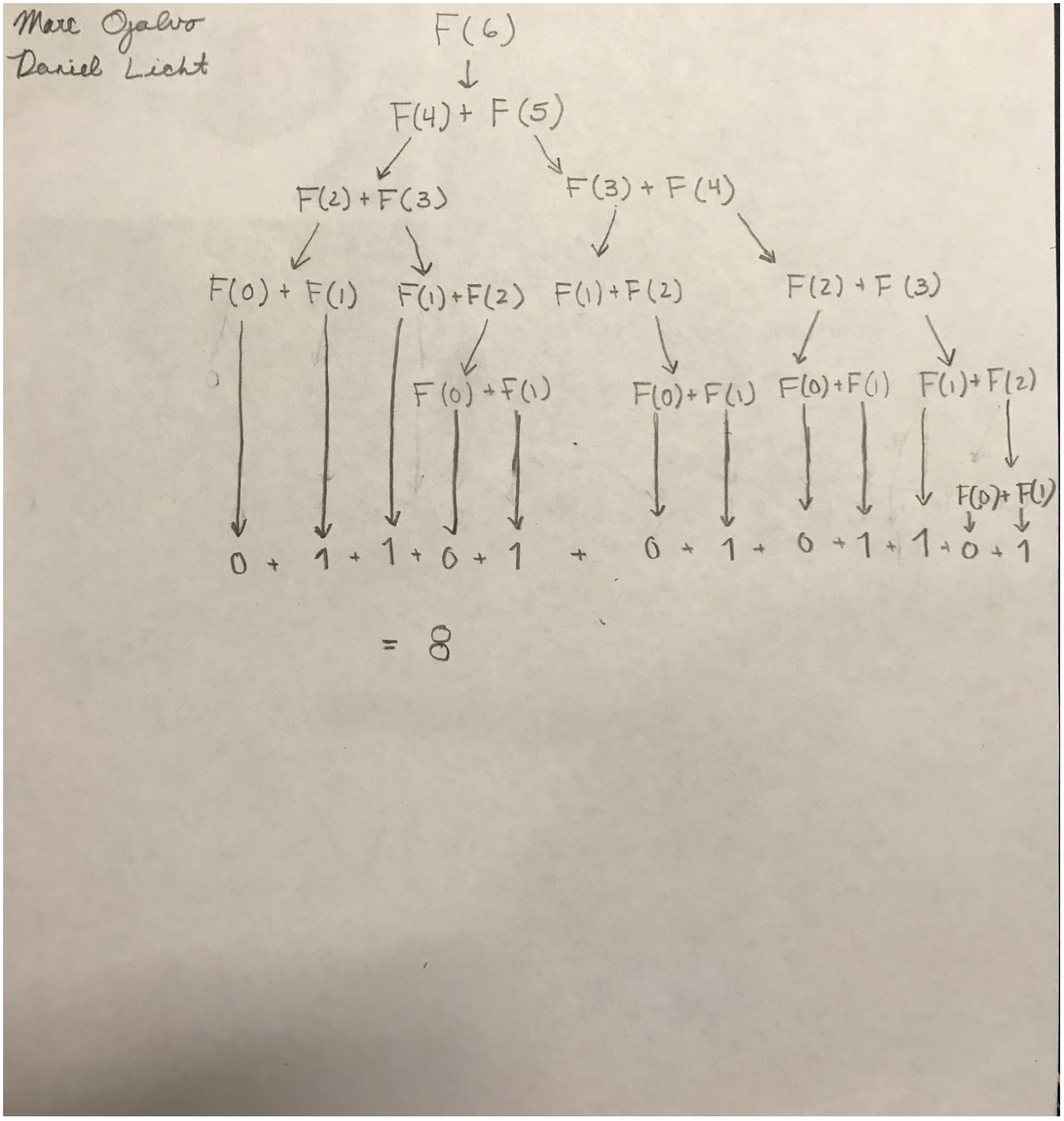
Marc Ojalvo and Daniel Licht
CMPS 1500
Thursday 3:30-4:45
Lab 7 Part 2
04/06/2018



n	Recursive (F(n))	Iterative (f(n))
10	0.00013589859008789	0.0000391006469726562
20	0.00473618507385253	0.00004506111145019531
30	0.391796827316284	0.00003933906555175781
40	48.5304698944091	0.000044107437133789

The iterative function is much more efficient than the recursive function. The iterative function does not take significantly more time as the value of n increase from 10 to 40 whereas the recursive function takes much more time with every increase of n. Although the run time for the iterative Fibonacci number function did not really change, the recursive function's run time increased a lot because each call of the function requires calls of smaller functions until it reaches F(0) and/or F(1). This means that the higher number calls to F(n) require deeper recursion while the iterative function only uses one for loop regardless of the value of n, and thus recursion has longer run time.

```
import time

def F(n):
    """ Returns the nth Fibonacci number in the Fibonacci
        sequence using recursion.
        Args:
            n (int): place of number to return in Fibonacci sequence
        Returns:
            int: nth Fibonacci number
    """
    if n == 0 or  n == 1:       #base case
        return n
    return F(n - 2) + F(n - 1)  #recursive step

def f(n):
    """ Returns the nth Fibonacci number in the Fibonacci
        sequence using iteration.
        Args:
            n (int): place of number to return in Fibonacci sequence
        Returns:
            int: nth Fibonacci number
    """
    number = 0
```

```
    number1 = 1
    for i in range(n):
        number += number1
        number1 = number - number1
    return number

t1 = time.time()
t1_nice = time.ctime(t1)
f(n)                                #where n is the Fibonacci number being tested
t2 = time.time()
t2_nice = time.ctime(t2)
total_time = t2 - t1
print(total_time)
```