Homework 1:

due September 4, 2020

Individual contributions only, submit via D2L, only typeset solutions in pdf-format are accepted

In this homework, we evaluate the performance of a recursive version of Fibonacci that

The following Python code calculates recursively the Fibonacci numbers, defined by

$$f_n = \begin{cases} 0 & \text{for } n = 0 \\ 1 & \text{for } n = 1 \\ f_{n-1} + f_{n-2} & \text{for } n \geq 2 \end{cases} .$$

```
def rec_fib(n):
    if n < 2:
        return n
    else:
        return rec_fib(n-1)+rec_fib(n-2)
```

If you implement this code and run it, you will find that for even moderately large arguments n, this implementation will take too long. On my machine, I can barely calculate `rec_fib(35)`, and the situation does not become much better if I use C++ instead of Python. The reason is that for each increment of n, I have two recursive calls, which often create additional recursive calls. In fact, we will show that the number of recursive calls increases very much like the Fibonacci sequence itself.

If the argument n is 0 or 1, there is no recursive call. If the argument is 2, then there are recursive calls with arguments 0 and 1 and no further calls. If the argument is 3, then there will be recursive calls with arguments 2 and 1, the former creating 2 more recursive calls. Thus, we have

Argument	Recursive Calls r_n
0	$r_0 = 0$
1	$r_1 = 0$
2	$r_2 = 2$
3	$r_3 = r_2 + r_1 + 2 = 4$

Develop a recurrence relation for the number of recursive calls r_n. Then **prove by induction** that $r_n = -2 + f_{n-1} + f_n + f_{n+1}$ for $n > 1$.