Patrick Braun, Class Group 3, Xinyi Xu

Homework 3

Instructions: Please submit your solutions via Gradescope by **Friday, 11 February 2022, 10:00am**. Make sure your name, your class group number, and the name of your class teacher is put on **every page** of your submission. Your submission should, ideally, be a **PDF** file.

Exercise 3.1: Fibonacci numbers, recursively **4 pts**

The Fibonacci numbers are a recursively-defined sequence of numbers, which arise in a surprising variety of real-world phenomena. The nth Fibonacci number is usually denoted by F_n and has the following recursive definition:

$$\begin{aligned} F_1 &= 1, \\ F_2 &= 1, \\ F_n &= F_{n-1} + F_{n-2}, \quad \text{for } n > 2. \end{aligned}$$

(a) Write Python code that, given $n \geq 1$ as an argument, implements the natural recursive algorithm for computing the nth Fibonacci number F_n.

(b) Measure the time it takes for computing the nth Fibonacci number F_n for small values of n (up to say 40 or 45): for example, using the code snippet `stopwatch.py` provided on the course's Moodle page. Use this to explore the ratio between the running times for two consecutive values of n. What do you observe?

(c) Let $a, b > 0$ be positive constants. Then, the running time of the recursive algorithm is well captured by the following recurrence:

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + b, && \text{for } n \geq 3, \text{ and} \\ T(n) &= a && \text{for } n = 1, 2. \end{aligned}$$

Use induction to show that

$$T(n) = (a+b)F_n - b, \qquad \text{for all } n \geq 1.$$

(d) Assume $a = b = 1$. The number $\phi = (1 + \sqrt{5})/2 \approx 1.618$ is known as the Golden ratio. Use Binet's formula for the nth Fibonacci number F_n, given as

$$F_n = \frac{1}{\sqrt{5}} \left(\phi^n - (-\phi)^{-n} \right),$$

to argue that $T(n) = \Omega(\phi^n)$.

Exercise 3.2: Fibonacci numbers, iteratively **3 pts**

The natural recursive algorithm for computing the nth Fibonacci number F_n has exponential running time. From a time complexity perspective, that is really terrible. From a practical perspective, this means that you will not be able to compute the nth Fibonacci number F_n even for moderately sized values of n using the recursive algorithm.

Luckily, there is a more clever iterative algorithm for computing the nth Fibonacci number that runs in linear time.

(a) Describe this algorithms in words.

(b) Implement this algorithm in Python.

(c) Argue, using big-O notation, that the running time of your algorithm is $O(n)$.

Exercise 3.3: Big O-notation and the sum rule **3 pts**

(a) Show that, if $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then

$$f(n) = f_1(n) + f_2(n) = O(g_1(n) + g_2(n)).$$

(b) For functions in part (a), do we also have $f(n) = O(\max\{g_1(n), g_2(n)\})$?

(c) Is it also true that if $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$, then

$$f(n) = f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))?$$

Exercise 3.1: Fibonacci numbers, recursively **4 pts**

The Fibonacci numbers are a recursively-defined sequence of numbers, which arise in a surprising variety of real-world phenomena. The nth Fibonacci number is usually denoted by F_n and has the following recursive definition:

$$F_1 = 1,$$
$$F_2 = 1,$$
$$F_n = F_{n-1} + F_{n-2}, \quad \text{for } n > 2.$$

(a) Write Python code that, given $n \geq 1$ as an argument, implements the natural recursive algorithm for computing the nth Fibonacci number F_n.

(a)

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

(b) Measure the time it takes for computing the nth Fibonacci number F_n for small values of n (up to say 40 or 45): for example, using the code snippet stopwatch.py provided on the course's Moodle page. Use this to explore the ratio between the running times for two consecutive values of n. What do you observe?

(b)

```python
# Timing code
times = []
for i in range(20, 35):
    start1 = time.time()
    fib_recursive(i)
    end1 = time.time()
    elapsed1 = end1 - start1
    times.append(elapsed1)


for i in range(14):
    print(times[i+1]/times[i])
```

```
1.94519906323185
1.5148085721165423
1.417660149419806
1.5864214834333128
1.9270240661554228
1.4614608740303325
1.7367991768307987
1.6410204538722193
1.567691914833665
1.6603449292701216
1.624667627417947
1.6900375842000603
1.5566168380667011
1.5787463694959698
```

Ratio of increasing F_n

We can see that the ratio between the times of F_n is somewhere around 1.6 (in particular is ϕ)
Hence, the running time is scaling exponentially.

(c) Let $a, b > 0$ be positive constants. Then, the running time of the recursive algorithm is well captured by the following recurrence:

$$T(n) = T(n-1) + T(n-2) + b, \qquad \text{for } n \geq 3, \text{ and } (1)$$
$$T(n) = a \qquad \text{for } n = 1, 2.$$

Use induction to show that

$$T(n) = (a+b)F_n - b, \qquad \text{for all } n \geq 1.$$

Using strong induction

Base Case: $T(1) = (a+b)F_1 - b = a+b-b = a$ ✓

$T(2) = (a+b)F_2 - b = a+b-a = a$ ✓

Inductive step Let $k \in \mathbb{N}$ and $k > 2$

Suppose $T(n) = (a+b)F_n - b \quad \forall t \leq k$

Want to show $T(k+1)$. We know it holds for $k, k-1$

so $T(k) = (a+b)F_k - b$

$T(k-1) = (a+b)F_{k-1} - b$ (*)

$T(k+1) = T(k) + T(k-1) + b$ by (1)

$= (a+b)F_k - b + (a+b)F_{k-1} - b + b$ by (*)

$= (a+b)(F_k + F_{k-1}) - b$

$= (a+b)F_{k+1} - b$ by definition of F

so inductive step holds.

So by strong induction holds for all $n \in \mathbb{N}$ ∎

(d) Assume $a = b = 1$. The number $\phi = (1+\sqrt{5})/2 \approx 1.618$ is known as the Golden ratio. Use Binet's formula for the nth Fibonacci number F_n, given as

$$F_n = \frac{1}{\sqrt{5}}\left(\phi^n - (-\phi)^{-n}\right),$$

to argue that $T(n) = \Omega(\phi^n)$.

From (c) $T(n) = (a+b)F_n - b \quad \forall n \geq 1$

as $a = b = 1$

$T(n) = 2F_n - 1$ (by Binet's)

$T(n) = 2\frac{1}{\sqrt{5}}\left(\phi^n - \frac{1}{(-\phi)^n}\right) - 1 \approx \frac{2}{\sqrt{5}}\left(\phi^n - \frac{1}{(-0.618)^n}\right) - 1$

$= \frac{2}{\sqrt{5}}\phi^n - \underbrace{\frac{2}{\sqrt{5}}(-\phi)^{-n}}_{\to 0 \text{ as } n \to \infty} - 1 \geq \frac{1}{\sqrt{5}}\phi^n$ for $n > 100$ for example

So $T(n) = \Omega(\phi^n)$

Exercise 3.2: Fibonacci numbers, iteratively **3 pts**

The natural recursive algorithm for computing the nth Fibonacci number F_n has exponential running time. From a time complexity perspective, that is really terrible. From a practical perspective, this means that you will not be able to compute the nth Fibonacci number F_n even for moderately sized values of n using the recursive algorithm.

Luckily, there is a more clever iterative algorithm for computing the nth Fibonacci number that runs in linear time.

(a) Describe this algorithms in words.

(a) We can calculate F_1, F_2 and then F_3 ... up to F_n.
To do we simply use the formula $F_n = F_{n-1} + F_{n-2}$.
Then we keep iterating till we get to F_n.

(b) Implement this algorithm in Python.

```
1 def fibonacci_iter(n):         O(1)
2     if n <= 2:                 O(1)
3         return 1               O(1)
4     ls = [1, 1]                O(1)
5     for i in range(n-2):       O(n-2)
6         ls.append(ls[-1] + ls[-2])   O(n-2)
7     return ls[-1]              O(1)
```

(c) Argue, using big-O notation, that the running time of your algorithm is $O(n)$.

Everything except line 5, 6 is $O(1)$. Line 5, 6 run $n-2$ times. So adding up we get
$O(1) + \ldots O(1) + O(n-2) + O(n-2) + O(1) = O(n)$

Exercise 3.3: Big O-notation and the sum rule

(a) Show that, if $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then

$$f(n) = f_1(n) + f_2(n) = O(g_1(n) + g_2(n)).$$

(a) $f_1(n) = O(g_1(n)) \iff \exists c_1, n_1$ s.t $f_1(n) \leq c_1 g_1(n)$ $\forall n \geq n_1$

$f_2(n) = O(g_2(n)) \iff \exists c_2, n_2$ s.t $f_2(n) \leq c_2 g_2(n)$ $\forall n \geq n_2$.

Then if $n \geq \max\{n_1, n_2\}$,

$f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n) \leq \max\{c_1, c_2\} (g_1(n) + g_2(n))$

$f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$

(b) For functions in part (a), do we also have $f(n) = O(\max\{g_1(n), g_2(n)\})$? (yes)

Let $c = \max\{c_1, c_2\}$

Again $f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n)$

$\leq c(g_1(n) + g_2(n))$

$\leq c(2\max\{g_1(n), g_2(n)\})$

$\leq 2c \max\{g_1(n), g_2(n)\}$

so $f_1(n) + f_2(n) = O(\max\{g_1(n), g_2(n)\})$

(c) Is it also true that if $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$, then

$$f(n) = f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))?$$

(c) $f_1(n) = \Omega(g_1(n)) \iff \exists c_1, n_1$ s.t $f_1(n) \geq c_1 g_1(n)$ $\forall n \geq n_1$

$f_2(n) = \Omega(g_2(n)) \iff \exists c_2, n_2$ s.t. $f_2(n) \geq c_2 g_2(n) \ \forall n \geq n_2$

Then if $n \geq \max\{n_1, n_2\}$

$f_1(n) + f_2(n) \geq c_1 g_1(n) + c_2 g_2(n) \geq \min\{c_1, c_2\}(g_1(n) + g_2(n))$

so $f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$