CMPS 2200 Assignment 1

Name: Miranda Díaz

In this assignment, you will learn more about asymptotic notation, parallelism, functional languages, and algorithmic cost models. As in the recitation, some of your answer will go here and some will go in `main.py`. You are welcome to edit this `assignment-01.md` file directly, or print and fill in by hand. If you do the latter, please scan to a file `assignment-01.pdf` and push to your github repository.

1. (2 pts ea) **Asymptotic notation**

- 1a. Is $2^{n+1} \in O(2^n)$? Why or why not? . Yes.

A constant multiple of 2^n is always greater than 2^{n+1}

$2^{n+1} = 2 \cdot 2^n \leq c2^n$, $c \geq 2$

- 1b. Is $2^{2^n} \in O(2^n)$? Why or why not?

No.

$f(n) = 2^{2n}$

$g(n) = 2^n$

def of $w = f(n) > c \cdot g(n)$ for all $n \geq k$

$2^{2n} > c \cdot 2^n$ for all $n \geq k$

$= (2^n)^2 > c \cdot 2^n$ for all $n \geq k$

$= \cancel{(2^n)}(2^n) > c \cdot \cancel{2^n}$ for all $n \geq k$

$2^n > c$ for all $n \geq k$

$\log 2n > \log c$ for all $n \geq k$

$n > \log_2 c$ for all $n \geq \log_2 c + 1$

Thus, $2^{2n} \in w(2^n)$, so no.

$2^{2n} \notin O(2^n)$

- 1c. Is $n^{1.01} \in O(\log^2 n)$?

$f(n) = n^{1.01}$ No

$g(n) = \log^2 n$

Only if there exist positive constants c and n_0 such that $n^{1.01} \leq c \log^2 n$ for all $n \geq n_0$

- 1d. Is $n^{1.01} \in \Omega(\log^2 n)$? yes

There exist positive constant c such that it lies above $c \cdot g(n)$ for a sufficienty large n

- 1e. Is $\sqrt{n} \in O((\log n)^3)$? No

There does not exist a positive constant c such that lies between 0 and $c \cdot g(n)$, for sufficienly large n

- 1f. Is $\sqrt{n} \in \Omega((\log n)^3)$? yes

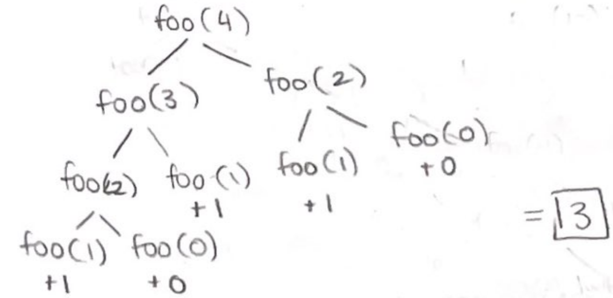
$f(n) = n^{1/2}$

$g(n) = (\log n)^3$

There exits a positive constant c that lies above $c(g(n))$ for a sufficienty large n

Example

0, 1, 1, 2



2. **SPARC to Python**

Consider the following SPARC code of the Fibonacci sequence, which is the series of numbers where each number is the sum of the two preceding numbers. For example, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610 ...

```
foo x =
  if x ≤ 1 then
    x
  else
    let (ra, rb) = (foo (x − 1)) , (foo (x − 2)) in
      ra + rb
    end.
```

10 - 1
9 + 8 =

- 2a. (6 pts) Translate this to Python code – fill in the `def foo` method in `main.py`
- 2b. (6 pts) What does this function do, in your own words?

This is a recursive problem, it will get broken down into smaller pieces until it reaches the base case. In foo, we're checking the base case first. The next instruction is to return the sum of the values that happens after you call the function with the two values prior to x. For example, to calculate foo(9), foo calls on itself 9 times, until it reaches the base case, then just adding the values, 1 and 0 however many times is fit.

3. **Parallelism and recursion**

Consider the following function:

```
def longest_run(myarray, key)
    """
    Input:
      `myarray`: a list of ints
      `key`: an int
    Return:
      the longest continuous sequence of `key` in `myarray`
    """
```

E.g., `longest_run([2,12,12,8,12,12,12,0,12,1], 12) == 3`

- 3a. (7 pts) First, implement an iterative, sequential version of `longest_run` in `main.py`.
- 3b. (4 pts) What is the Work and Span of this implementation?