CMPS 2200 Assignment 1

Name:_________________________

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? .
  . If $|f(n)| \leq M * g(n)$ then $f(n)\ \epsilon\ O(g(n))$
  . $f(n) = 2^{n+1},\ g(n) = 2^n$
  . $f(n) = 2^{n+1} = 2 * 2^n = 2 * g(n)$
  . Therefore, $f(n)\ \epsilon\ O(g(n))$

- 1b. Is $2^{2^n} \in O(2^n)$? Why or why not?
  . If $|f(n)| \leq M * g(n)$ then $f(n)\ \epsilon\ O(g(n))$
  . $f(n) = 2^{2^n},\ g(n) = 2^n$
  . $f(n) = 2^n \log 2, g(n) = n \log 2$
  . $2^n \log 2\ \nleq n \log 2$
  . Therefore, $f(n)\ \notin\ O(g(n))$

- 1c. Is $n^{1.01} \in O(\log^2 n)$?
  . If $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} [0,\infty): f(n)\ \epsilon\ O(g(n)) \\ (0,\infty):\ f(n)\ \epsilon\ \Theta(g(n)) \\ (0,\infty]: f(n)\ \epsilon\ \Omega(g(n)) \end{cases}$
  . $f(n) = n^{1.01}, g(n) = log^{\,2}\, n$
  . $\lim\limits_{n\to\infty} \frac{n^{1.01}}{log^2 n} = \lim\limits_{n\to\infty} \frac{(n^{1.01})'}{(log^2 n)'} = \lim\limits_{n\to\infty} \frac{1.01 * n^{1.01}}{2 \log n} = \infty$
  . so $f(n)\ \notin\ O(g(n))$

- 1d. Is $n^{1.01} \in \Omega(\log^2 n)$?
  . $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} [0,\infty): f(n)\ \epsilon\ O(g(n)) \\ (0,\infty):\ f(n)\ \epsilon\ \Theta(g(n)) \\ (0,\infty]: f(n)\ \epsilon\ \Omega(g(n)) \end{cases}$
  . $f(n) = n^{1.01}, g(n) = log^{\,2}\, n$
  . $\lim\limits_{n\to\infty} \frac{n^{1.01}}{log^2 n} = \lim\limits_{n\to\infty} \frac{(n^{1.01})'}{(log^2 n)'} = \lim\limits_{n\to\infty} \frac{1.01 * n^{1.01}}{2 \log n} = \infty$
  . so $f(n)\ \epsilon\ \Omega(g(n))$

- 1e. Is $\surd n\_ \in O((\log n)^3)$?

  . $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} [0,\infty): f(n) \, \epsilon \, O(g(n)) \\ (0,\infty):\ f(n) \, \epsilon \, \Theta(g(n)) \\ (0,\infty]: f(n) \, \epsilon \, \Omega(g(n)) \end{cases}$

  . $f(n) = \sqrt{n}, g(n) = (\log\ n)^3$

  . $\lim_{n\to\infty} \frac{\sqrt{n}}{(\log\ n)^3} = \lim_{n\to\infty} \frac{\sqrt{n}'}{\sqrt{(\log\ n)^6}'} = \lim_{n\to\infty} \frac{\frac{1}{2\sqrt{n}}}{\frac{6*(\log n)^5}{2*\sqrt{(\log n)^6}}} = \infty$

  . therefore $f(n) \ \notin \ O(g(n))$

- 1f. Is $\surd n\_ \in \Omega((\log n)^3)$?

  . $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} [0,\infty): f(n) \, \epsilon \, O(g(n)) \\ (0,\infty):\ f(n) \, \epsilon \, \Theta(g(n)) \\ (0,\infty]: f(n) \, \epsilon \, \Omega(g(n)) \end{cases}$

  . $f(n) = \sqrt{n}, g(n) = (\log\ n)^3$

  . $\lim_{n\to\infty} \frac{\sqrt{n}}{(\log\ n)^3} = \lim_{n\to\infty} \frac{\sqrt{n}'}{\sqrt{(\log\ n)^6}'} = \lim_{n\to\infty} \frac{\frac{1}{2\sqrt{n}}}{\frac{6*(\log n)^5}{2*\sqrt{(\log n)^6}}} = \infty$

  . therefore $f(n) \, \epsilon \, \Omega(g(n))$

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.
```

- 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?

```
def foo(n):
  if n ≤ 1:
    return n
  else:
    ra, rb = foo(n-1), foo(n-2)
    return ra + rb
```

This Python code defines a function that takes an integer n as input and returns the nth number in the Fibonacci sequence. If n is less than or equal to 1, the function simply returns n. This serves as the base case so that numbers less than 1 do not get included. Otherwise, it recursively computes the (n-1)th and (n-2)th numbers in the sequence using the function, and sums them to get the nth number.

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?

.
. Work is O(n)
. Span is O(n)
.
.
.
.
.
.

- 3c. (7 pts) Next, implement a longest_run_recursive, a recursive, divide and conquer implementation. This is analogous to our implementation of sum_list_recursive. To do so, you will need to think about how to combine partial solutions from each recursive call. Make use of the provided class Result.
- 3d. (4 pts) What is the Work and Span of this sequential algorithm?

  .
  . Work is O(n log n)
  . Span is O(log n)

.
.
.
.
.
.
.
.

- 3e. (4 pts) Assume that we parallelize in a similar way we did with sum_list_recursive. That is, each recursive call spawns a new thread. What is the Work and Span of this algorithm?

.

. Work is O(n log n)
. Span is O(log n)

.
.
.
.
.
.