# CMPS 2200 Assignment 1

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. Asymptotic notation

- 1a. Is  $2^{n+1} \in O(2^n)$ ? Why or why not? .
  - .
  - .
- 1b. Is  $2^{2^n} \in O(2^n)$ ? Why or why not?
  - .
  - .
  - •
- 1c. Is  $n^{1.01} \in O(\log^2 n)$ ?
- •
- .
- 1d. Is  $n^{1.01} \in \Omega(\log^2 n)$ ?
  - .
  - .
- 1e. Is  $\sqrt{n} \in O((\log n)^3)$ ?
  - .
- .
- 1f. Is  $\sqrt{n} \in \Omega((\log n)^3)$ ?
- .
- .
- 1g. Consider the definition of "Little o" notation:

 $g(n) \in o(f(n))$  means that for **every** positive constant c, there exists a constant  $n_0$  such that  $g(n) \le c \cdot f(n)$  for all  $n \ge n_0$ . There is an analogous definition for "little omega"  $\omega(f(n))$ . The distinction between o(f(n)) and O(f(n)) is that the former requires the condition to be met for **every** c, not just for some c. For example,  $10x \in o(x^2)$ , but  $10x^2 \notin o(x^2)$ .

.

**Prove** that  $o(g(n)) \cap \omega(g(n))$  is the empty set.

·
·
·
·

.

## 2. Python to SPARC

Recall the sum\_list\_recursive\_parallel function from lecture 2. Specify our implementation in SPARC here.

.

. . . . . . .

.

## 3. SPARC to Python

Consider the following SPARC code:

```
\begin{array}{l} foo\ x=\\ \text{ if }\ x\leq 1\ \text{ then }\\ x\\ \text{ else }\\ \text{ let }(ra,rb)=(foo\ (x-1))\ ,\ (foo\ (x-2))\ \text{ in }\\ ra+rb\\ \text{ end. } \end{array}
```

- 3a. Translate this to Python code fill in the def foo method in main.py
- 3b. What does this function do, in your own words?

. . . . . .

.

#### 4. 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
  • 4a. First, implement an iterative, sequential version of longest_run in main.py.
```

- 4b. What is the Work and Span of this implementation?

- 4c. 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.
- 4d. What is the Work and Span of this sequential algorithm?

• 4e. 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?