CMPS 2200 Assignment 1

Name: Haowen Weng.

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? .

  Yes. $2^{n+1} = 2 \cdot 2^n$

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

  $2^{2^n} = 4^n$ the image of 2^{2^n} is above 2^n

  No.

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

  No $n^{1.01}$ grows faster than $\log^2 n$

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

  Yes.

- 1e. Is $\sqrt{n} \in O((\log n)^3)$?

  No

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

  Yes

- 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 $g(n) \leq c \cdot f(n)$ for all $n \geq n_0$. 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.

Suppose $o(g(n)) \cap \omega(g(n))$ is not the empty set.

let $f(n) = o(g(n)) \cap \omega(g(n))$

$$o(g(n)) = \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

$$\omega(g(n)) = \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

$\Rightarrow$ as $n \to \infty$, these two equation cannot both be true.

$\therefore$ $f(n)$ doesn't exist.

$\therefore$ $o(g(n)) \cap \omega(g(n))$ is the empty set.

2. **SPARC to Python**

Consider the following SPARC code:

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

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

The function calculated the nth fibonacci 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. First, implement an iterative, sequential version of `longest_run` in `main.py`.
- 3b. What is the Work and Span of this implementation?

Work: $3+4\ O(n)$ Span: $2+\theta(\lg n)$