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

Proof by Contradiction

$f(n) \in o(g(n)) \cap \omega(g(n))$

Then, for every positive constant c_1 there exists a constant n_0 such that

$f(n) \leq c_1(g(n))$ for all $n \geq n_0$

For every positive constant c_2, there exists a constant n_0 such that

$c_2(g(n)) \leq f(n)$ for all $n \geq n_0$.

If $c_1 = 1$ and $c_2 = 5$, then $5(g(n)) \leq f(n) \leq g(n)$.

This is impossible.

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?

This function determines the Xth term in the Fibonacci sequence.

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: $O(n)$

Span: $O(n)$

CMPS 2200 Assignment 1

Name: Naomi Stoner

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
  - If $c=3$ and $n_0=-1$,
  - $2^{n+1} \leq 3 \cdot 2^n$
  -
  -

- 1b. Is $2^{2^n} \in O(2^n)$? Why or why not? No
  - $\ln(2^{2^n}) \leq \ln(2^n)$ for $n_0 = 0$
  - $2^n \leq n$? No.
  -

- 1c. Is $n^{1.01} \in O(\log^2 n)$? No
  - If $n \geq 1$, then $n^{1.01} \geq \log^2 n$
  - Furthermore, $\lim_{n \to \infty} \left(\frac{n^{1.01}}{\log^2 n}\right) = 0$
  -
  -

- 1d. Is $n^{1.01} \in \Omega(\log^2 n)$? Yes
  - $n^{1.01} \geq c \cdot \log^2 n$
  - for $c=1$ and $n_0 = 1$ this is true
  - Furthermore, $\lim_{n \to \infty} \left(\frac{n^{1.01}}{\log^2 n}\right) = \infty$

- 1e. Is $\sqrt{n} \in O((\log n)^3)$? Yes
  - for $c = 12$ and $n_0 = 2$
  - $\sqrt{n} \leq 12(\log n)^3$
  -
  -

- 1f. Is $\sqrt{n} \in \Omega((\log n)^3)$? No
  - There is no value for which $c \cdot \sqrt{n}$ asymptotically
  - dominates $(\log n)^3$
  -

- 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 $g(n) \leq c \cdot f(n)$ 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 $10x^2 \notin o(x^2)$.

.

- 3c. 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. What is the Work and Span of this sequential algorithm?

$W(n)$ is equal to the ~~number of~~ nodes, as each node has $O(1)$ work.
Therefore, $W(n) = n\log n$, as the work is balanced
$S(n) = 2\log_2(n) \in O(\log_2 n)$

- 3e. 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?

longest_helper completes in $O(1)$ time, so:
$W(n) = 2n \in O(n)$

$S(n) = 2(\log_2(n)) \in O(\log_2 n)$