CMPS 2200 Assignment 1

Name: Raphael Deyssin

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? .

there have to be a constant C: $2^{n+1} \leq C \cdot 2^n$

this constant exists, and its 2. So yes, $2^{n+1} \in O(2^n)$

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

$2^{2^n} \leq C \cdot 2^n \rightarrow \log \rightarrow 2^n \leq \log C \cdot n \rightarrow$

$\frac{2^n}{n} \leq \log C \rightarrow$ there is no constant that can make this true for $n > 0$ so no

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

$n^{1.01} \leq C \cdot \log^2 n$.

$\frac{1}{\log n} \cdot n^{1.01} \leq C \cdot \log n$ as $n \to \infty$ $\frac{1}{\log(n)} \to 0 \rightarrow n^{1.01} \leq C \cdot \log(n) \rightarrow$ false. there is no C that allows $\log(n) > n$

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

$n^{1.01} \geq C \cdot \log^2 n$ by the same logic as above. yes. $n^{1.01}$ will always dominate $\log^2 n$

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

$\sqrt{n} \leq C \cdot (\log n)^3 \rightarrow \frac{\sqrt{n}}{(\log(n))^3} \leq C \rightarrow \lim_{n \to \infty} \left(\frac{\sqrt{n}}{(\log(n))^3} \leq C \right) \rightarrow$

$\infty \leq C \rightarrow \infty$ will never be less than a constant therefore $\sqrt{n}$ isn't $O(\log^3 n)$

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

$\sqrt{n} \geq C \cdot (\log n)^3 \rightarrow \frac{\sqrt{n}}{(\log n)^3} \geq C \rightarrow$ if we take the limit of both sides $n \to \infty$ we will get $\infty \geq c$. So yes $\sqrt{n}$ is $\Omega((\log n)^3$

- 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 $g(n) \leq c \cdot f(n)$ 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.

assume $o(g(n)) \cap \omega(g(n))$ is not empty

- $o(g(n)) \rightarrow g(n) < f(n) \rightarrow \; < \frac{f(n)}{g(n)}$
- $\omega(g(n)) \rightarrow g(n) > f(n) \rightarrow \; > \frac{f(n)}{g(n)}$

We know that for every c, it is impossible for these two statements to hold true. therefore, our initial assumption must be wrong, and

$o(g(n)) \cap \omega(g(n)) = \{\emptyset\}$

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 is the Fibonacci code

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`

(1 2 3 4 5 6 7 8 9 10)

3a. First, implement an iterative, sequential version of `longest_run` in `main.py`.

- 3b. What is the Work and Span of this implementation?

We are summing n elements over a list of n So the work is proportional to the length of the list Hence $W(n) \in O(n)$

the span is also $\in O(n)$

- 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?

Work for each level = $O(n)$

So $W(n) = O(n \log n)$

Span = $O(\log n)$

- 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?

the work would remain unchanged but the Span would be $O(n)$

So work = $O(n \log n)$

Span = $O(n)$