CMPS 2200 Assignment 1

Name: Hailey Dusablon

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? .
  - Asymptotic Dominance: g(n)≤ c * f(n) for all n > n_0.
  - 2^{n+1} will always have a larger value than 2^n. Therefore, 2^n does not asymptotically dominate 2^{n+1}.
  - .
  - .
- 1b. Is $2^{2^n} \in O(2^n)$? Why or why not?
  - 2^{2^n} grows at a faster rate than 2^n.
  - Because of this, there is no c where g(n)≤ c * f(n) for all n > n_0.
  - 2^n does not asymptotically dominate 2^{2^n} .
  - .
  - .
- 1c. Is $n^{1.01} \in O(\log^2 n)$?
  - $n^{1.01}$ is functionally linear, and linear functions dominate logarithmic functions such as log^2 n
  - log^2 n does not asymptotically dominate n^1.01.
  - .
  - .
- 1d. Is $n^{1.01} \in \Omega(\log^2 n)$?
  - yes- this is the opposite of previous question, which was false.
  - .
  - .
  - .
- 1e. Is $\sqrt{n} \in O((\log n)^3)$?
  - √n is grows at a slightly faster rate than (logn)^3. Since it starts at a higher value, there is no value of n where g(n) ≤ f(n).
  - (logn)^3 does not asymptotically dominate sq rt(n).
  - .
  - .
- 1f. Is $\sqrt{n} \in \Omega((\log n)^3)$?
  - Yes- this is opposite of previous question, which is false.
  - .
  - .
  - .
- 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.

- o(g(n)) = g(n) < c * f(n) , ω(g(n)) = g(n) > c * f(n)
- f(n)*c > g(n) > f(n)*c
- f(n)c cannot be greater than and less than g(n) at the same time.
- Therefore, the intersection of little o and little omega is the empty set by contradiction.
- .
- .
- .
- .
- .
- .
- .
- .

2. **SPARC to Python**

Consider the following SPARC code:

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

- 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 code recursively defines the xth number 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?

- .
- .
- .
- .

- 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) = O(n)
  - S(n) = O(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?

- W(n) = O(n)
- S(n) = O($\log_2$n)