CMPS 2200 Assignment 1

Name: Sydney Feldman

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. (2 pts ea) **Asymptotic notation**

- 1a. Is $2^{n+1} \in O(2^n)$? Why or why not? .
  - yes
  - there exists a c (2) and an n_0 (1), where
  - $2^{n+1} \leq c \cdot 2^n$ for $n \geq 1$
  - $2^2 \leq 2 \cdot 2^1 \Rightarrow 4 \leq 4$, $2^{3+1} \leq 2 \cdot 2^3 \Rightarrow 2^4 \leq 2^4$, both are true
- 1b. Is $2^{2^n} \in O(2^n)$? Why or why not?
  - no
  -
  - 2^{2^n} will not be $\leq$ to $c \cdot 2^n$ for some
  - value of c and all values $n \geq n_0$
  - $2^{2^1} \leq 2 \cdot 2^1$ is true, but $2^{2^2} \leq 2 \cdot 2^2$ is false.
- 1c. Is $n^{1.01} \in O(\log^2 n)$?
  - no
  - let $c=100$ and $n_0=100$
  - $100^{1.01} \leq 100 \cdot \log^2 100$ is true, but
  - $1000^{1.01} \leq 100 \cdot \log^2 100$ is false, so there are no values of c & n_0 that make this true for $n \geq n_0$
- 1d. Is $n^{1.01} \in \Omega(\log^2 n)$?
  - yes
  - let $c=1$, $n_0=2$
  - $2^{1.01} \geq 1 \cdot \log^2 2$ and $100^{1.01} \geq 1 \cdot \log^2 100$
  - are both true, showing that there are 2 constants c and n_0 that make $f(n) \geq g(n)$ for all $n \geq n_0$.
- 1e. Is $\sqrt{n} \in O((\log n)^3)$?
  - no
  -
  -
  -
- 1f. Is $\sqrt{n} \in \Omega((\log n)^3)$?
  - yes

2. **SPARC to Python**

Consider the following SPARC code of the Fibonacci sequence, which is the series of numbers where each number is the sum of the two preceding numbers. For example, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610 ...

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

- 2a. (6 pts) Translate this to Python code – fill in the `def foo` method in `main.py`
- 2b. (6 pts) What does this function do, in your own words?

This function takes in an input, if it is ≤ 1, it returns the value of the input. If not, it creates 2 new variables that each call the function. 1 of their inputs is (x-1) and the other is (x-2). Each time it calls itself, it returns the sum of the previous 2 numbers in the Fibonacci sequence. This continues until the input is ≤ 1. Then, the last call returns the correct result.

3. **Parallelism and recursion**

Consider the following function:

```python
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. (7 pts) First, implement an iterative, sequential version of `longest_run` in `main.py`.
- 3b. (4 pts) What is the Work and Span of this implementation?

Work and Span are both O(n) because long-run here is a sequential algorithm, and it iterates through each node once.

- 3c. (7 pts) 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. (4 pts) What is the Work and Span of this sequential algorithm?

  $W(n) = O(n)$

  $S(n) = O(\log n)$

- 3e. (4 pts) 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?

  $W(n) = O(n)$

  $S(n) = O(\log n)$