CMPS 2200 Assignment 1

Name: Joe Wagner

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? .
  - $\exists c, n_0$ s.t. $0 \le f(n) \le cg(n)$ $\forall n \ge n_0$
  - $2^{n+1} \le c \cdot 2^n$
  - $\ln 2^{n+1} \le \ln(c \cdot 2^n)$
  - $n+1 \cdot \ln 2 \le \ln c + n \cdot \ln 2$
  - $n+1 \le \ln c + n$ → $1 \le \ln c$ Let $c \in \mathbb{R}$ s.t. $c > e$, then the inequality is always true, and the Big-O definition is true, so $2^{n+1} \in O(2^n)$ ■QED
- 1b. Is $2^{2^n} \in O(2^n)$? Why or why not?
  - $\exists c, n_0$ s.t. $0 \le 2^{2^n} \le c \cdot 2^n$ $\forall n \ge n_0$
  - $2^{2^n} \le c \cdot 2^n$
  - $2^{2 \cdot n} \le c \cdot 2^n$
  - $2^n \cdot 2^n \le c \cdot 2^n$
  - $2^n \le c$ ⇒ Contradiction, as 2^n is greater than a constant. Thus, $2^{2^n} \notin O(2^n)$ ■QED
- 1c. Is $n^{1.01} \in O(\log^2 n)$?
  - $\exists c, n_0$ s.t. $0 \le n^{1.01} \le c \cdot \log^2 n$ $\forall n \ge n_0$
  - $\lim_{n\to\infty} \frac{n^{1.01}}{\log^2 n} = \frac{\infty}{\infty}$
  - L'hopitals → $\lim_{n\to\infty} \frac{1.01n^{.01}}{\frac{2\log n}{n}} = \frac{2.02n^{.01}}{2\log n} = \frac{\infty}{\infty}$ → $\lim_{n\to\infty} \frac{.0202n^{-.99}}{\frac{2}{n}} = \frac{.0202n}{2n^{.99}} = \frac{\infty}{\infty}$ → $\lim_{n\to\infty} \frac{.0202}{1.98n^{-.01}} = \frac{.0202n^{.01}}{1.98} = \infty$

  Since the $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, $n^{1.01} \notin O(\log^2 n)$ ■QED
- 1d. Is $n^{1.01} \in \Omega(\log^2 n)$?
  - $\exists c, n_0$ s.t. $n^{1.01} \ge c \cdot \log^2 n$ $\forall n \ge n_0$
  - $\lim_{n\to\infty} \frac{n^{1.01}}{\log^2 n} = \frac{\infty}{\infty}$
  - $= \lim_{n\to\infty} \frac{1.01n^{.01}}{\frac{2\log n}{n}} = \frac{2.02n^{.01}}{2\log n} = \frac{\infty}{\infty}$ → $\lim_{n\to\infty} \frac{.0202n^{-.99}}{\frac{2}{n}} = \frac{.0202n}{2n^{.99}} = \frac{\infty}{\infty}$ → $= \lim_{n\to\infty} \frac{.0202}{1.98n^{-.01}} = \frac{.0202n^{.01}}{1.98} = \infty$

  Because the $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, $n^{1.01} \in \Omega(\log^2 n)$ ■QED

  $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \Rightarrow f(n) \in O(g(n)) \\ c>0 & \Rightarrow f(n) \in \Theta(g(n)) \\ \infty & \Rightarrow f(n) \in \Omega(g(n)) \end{cases}$
- 1e. Is $\sqrt{n} \in O((\log n)^3)$?
  - $\exists c, n_0$ s.t. $0 \le \sqrt{n} \le c \cdot (\log n)^3$ $\forall n \ge n_0$
  - $\lim_{n\to\infty} \frac{\sqrt{n}}{(\log n)^3} = \frac{\infty}{\infty}$
  - $= \lim_{n\to\infty} \frac{\frac{1}{2}n^{-1/2}}{3(\log n)^2 \frac{1}{n}} = \frac{\frac{1}{2}\sqrt{n}}{3\log^2 n} = \frac{\infty}{\infty}$ → $= \lim_{n\to\infty} \frac{\frac{1}{4}n^{-1/2}}{6\log n \frac{1}{n}} = \frac{\frac{1}{4}\sqrt{n}}{6\log n} = \frac{\infty}{\infty}$ → $= \lim_{n\to\infty} \frac{\frac{1}{8}n^{-1/2}}{6 \cdot \frac{1}{n}} = \frac{\frac{1}{8}n^{1/2}}{6} = \frac{\sqrt{n}}{48} = \infty$

  The $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, so thus $\sqrt{n} \notin O((\log n)^3)$ ■QED
- 1f. Is $\sqrt{n} \in \Omega((\log n)^3)$?
  - $\exists c, n_0$ s.t. $\sqrt{n} \ge c \cdot (\log n)^3$
  - $\lim_{n\to\infty} \frac{\sqrt{n}}{(\log n)^3} = \frac{\infty}{\infty}$
  - $= \lim_{n\to\infty} \frac{\frac{1}{2}n^{-1/2}}{3(\log n)^2 \frac{1}{n}} = \frac{\frac{1}{2}\sqrt{n}}{3\log^2 n} = \frac{\infty}{\infty}$ → $= \lim_{n\to\infty} \frac{\frac{1}{4}n^{-1/2}}{6\log n \frac{1}{n}} = \frac{\frac{1}{4}\sqrt{n}}{6\log n} = \frac{\infty}{\infty}$ → $= \lim_{n\to\infty} \frac{\frac{1}{8}n^{-1/2}}{6 \cdot \frac{1}{n}} = \frac{\frac{1}{8}n^{1/2}}{6} = \frac{\sqrt{n}}{48} = \infty$

  Since the $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, $\sqrt{n} \in \Omega((\log n)^3)$ ■QED

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 ...

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

- 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?

The function recursively implements the Fibonacci sequence. The base case is when x≤1, as this should result in 1 being returned. The recursive case adds the previous two Fibonacci numbers in the sequence, represented by ra and rb, and the sum of these is returned by ra+rb, and ra+rb is the sequence's next term. Running the function foo(x) will return the x+1th 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. (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?

- 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?
- 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?