CMPS 2200 Assignment 1

Name: Mackenzie Bookamer

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? . FALSE

  This is false because $n+1 > n$ for all values of n, meaning $2^{n+1} \notin O(2^n)$

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

  They have the same base, so let's compare their exponents.

  $f(n) = 2^n$, $g(n) = n$; $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, so $f(n) \in \Omega(g(n)) \Rightarrow f(n) \notin O(g(n))$

  FALSE

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

  let $f(n) = n^{1.01}$, $g(n) = \log^2 n$ $\Rightarrow \lim_{n\to\infty} \frac{n^{1.01}}{\log^2 n} = \infty$, so $n^{1.01} \in \Omega(\log^2 n)$, $n^{1.01} \notin O(\log^2 n)$

  FALSE

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

  Using the same limit as above, we know $n^{1.01} \in \Omega(\log^2 n) \Rightarrow$ TRUE

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

  let $f(n) = \sqrt{n}$, $g(n) = (\log n)^3$ $\Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{\sqrt{n}}{(\log n)^3} \Rightarrow$ $\sqrt{n}$ grows faster than $(\log n)^3$, so $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, so $\sqrt{n} \in \Omega((\log n)^3)$

  FALSE

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

  Reference the same limit as the previous question to see this is TRUE

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 ...

$$
\begin{array}{l}
\textit{foo}\ x = \\
\quad \texttt{if}\ \ x \le 1\ \ \texttt{then} \qquad n=0,1 \rightarrow \text{base cases} \\
\quad\quad x \\
\quad \texttt{else} \\
\quad\quad \texttt{let}\ \ (ra, rb) = \underbrace{(\textit{foo}\ (x-1))}_{n-1\ \text{term}}\ \ ,\ \ \underbrace{(\textit{foo}\ (x-2))}_{n-2\ \text{term}}\ \ \texttt{in} \\
\quad\quad\quad ra + rb \\
\quad\quad \texttt{end}.
\end{array}
$$

- 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 takes an integer input x and if x is less than 1 or equal to 1 it returns the value of x. otherwise, the function returns the sum of the 2 previous values.

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?

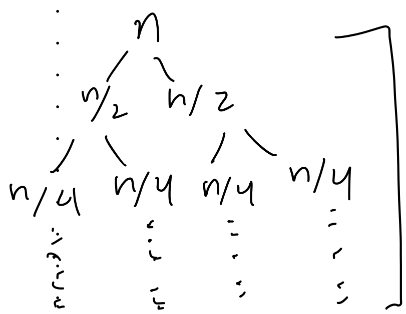
We run through the for loop for the value of len(mylist). Let's call this

value n. So, O(for-loop) = O(n). To find the max in list takes O(1) time.

Note both the work and span with be the same because every value of n is put into the for loop and n is a pre-determined size. So

$W, S \in O(n+1) \in O(n)$

- 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?



$W(n) = 2W(n/2) + n \rightarrow$ balanced!

$\therefore$ $\boxed{W(n) \in O(n \log n)}$

$\boxed{S(n) \in O(\log n)}$

- 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?

They will stay the same because f(n) is already running multiple branches, in essence, parallelized.

So, $\boxed{W(n) \in O(n \log n)}$

$\boxed{S(n) \in O(\log n)}$