CMPS 2200 Assignment 1

Name: Anna Schoeny

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? .

$\boxed{\text{Yes.}}$ Let $f(n) = 2^n$ and $g(n) = 2^{n+1}$. $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \frac{2^{n+1}}{2^n} = 2 \le c$

AKA $2^{n+1} \le 2^n \cdot c \rightarrow 2 \cdot 2^n \le 2^n \cdot c \rightarrow$ True where $c \ge 2$

$f(n)$ asymptotically dominates $g(n)$.

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

$2^{2^n} \le c \cdot 2^n$

$\log_2 2^{2^n} \le \log_2 c \cdot 2^n$

$2^n \le n \cdot \log_2 c$

$\boxed{\text{No.}}$ $\lim_{n\to\infty} \frac{2^{2^n}}{2^n} = \frac{g(n)}{f(n)}$ L'hopitals $\Rightarrow \lim_{n\to\infty} \frac{\ln^2(2) \cdot 2^{2^n} + n}{2^n \ln(2)} = \ln(2) \lim_{n\to\infty} \frac{2^{n+2^n}}{2^n}$

$\ln(2) \cdot \lim_{n\to\infty} (2^{2^n}) = \ln(2) \cdot \infty = \infty$ so c is unbounded and $f(n)$ doesn't asymp. dom. $g(n)$

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

$\boxed{\text{No.}}$ Let $f(n) = \log^2 n$ and $g(n) = n^{1.01}$

$\lim_{n\to\infty} \frac{n^{1.01}}{\log^2 n} = \infty$ and c cannot $\ge \infty$. $f(n)$ doesn't asymptotically dominate $g(n)$.

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

$\boxed{\text{Yes.}}$

$\lim_{n\to\infty} \frac{n^{1.01}}{\log^2 n} = \infty$ so $f(n) \in \Omega(g(n))$ where $f(n) = n^{1.01}$ and $g(n) = \log^2 n$ because $g(n)$ asymptotically dominates $f(n)$.

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

$\boxed{\text{No.}}$

$\lim_{n\to\infty} \frac{\sqrt{n}}{(\log n)^3} = \frac{f(n)}{g(n)} = \infty$ $g(n)$ does not asymptotically dominate $f(n)$.

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

$\boxed{\text{Yes}}$ $\lim_{n\to\infty} \frac{\sqrt{n}}{(\log n)^3} = \infty$ so $f(n) \in \Omega(g(n))$ where $f(n) = \sqrt{n}$, $g(n) = (\log n)^3$

- 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 $g(n) \le c \cdot f(n)$ for all $n \ge n_0$. 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.

Let $f(n) \in o(g(n))$. By definition,
$f(n) < c \cdot g(n)$.
Let $h(n) \in \omega(g(n))$. By definition,
$h(n) > c \cdot g(n)$.
Rewritten as $f(n) < c \cdot g(n) < h(n)$. In other words, there are no common elements between $o(g(n))$ and $\omega(g(n))$ because the statement inequality is not inclusive.

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?

foo is a Fibonacci recursion, which essentially means that the function determines the next value by summing the preceding 2 index's values. For example, foo(5) = 5 because the sequence starts at 0 or 1 and adds 1 (0, 1, 1, 2, 3, 5] so foo(5) ⇒ list[5] = 5 in essence.

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. First, implement an iterative, sequential version of `longest_run` in `main.py`. ✓
- 3b. What is the Work and Span of this implementation?

$W(n) = O(n)$ because it is iterative AKA linear

$S(n) = O(n)$

- 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?

$S(N) = 2S(n/2) + C_1$

$\boxed{W(n) \text{ and } S(n) = 2W(n/2) + C_1}$

because the leaf is split into 2 branches, each of which do 1/2 the work content and the combining is a constant value as no loops are used.

- 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?

Work wouldn't change from the previous question.
However, span would be

$S(N) = S(n/2) + C_1$ because span only follows one branch when parallelized because new threads are spawned instead.