CMPS 2200 Assignment 1

Name: Aidan Hussain

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? .

  Yes, the constant (1) has no effect on the asymptotic growth rate. when $c=3$, $c \cdot 2^n \geq 2^{n+1}$ for $n \geq 4$.

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

  No, 2^{2^n} has a greater asymptotic growth rate than 2^n (2^n grows faster than n). There is no c where $c \cdot 2^n \geq 2^{2^n}$ for all $n \geq n_0$.

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

  No, $n^{1.01}$ grows at a much faster rate and asymptotically dominates $O(\log^2 n)$.

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

  yes. when $c = 15$, $c \cdot \log^2 n \leq n^{1.01}$ for $n \geq 1571$.

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

  Yes. when $c=10$, $c \cdot (\log n)^3 \geq \sqrt{n}$ for $n \geq 4$.

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

  No, there exists no factor c for which $c(\log_2 n)^3 \leq \sqrt{n}$ for $n > n_0$.

- 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.

1. Let's assume a function, $f(n) \in o(g(n))$ and $\in \omega(g(n))$. We shall prove by contradiction that this is impossible and $o(g(n)) \cap \omega(g(n))$ is an empty set.
2. Since $f(n) \in \omega(g(n))$, we know, by definition, $g(n) \in o(f(n))$ since little omega is a lower bound.
3. We assumed in step 1, that $f(n) \in o(g(n))$. Substituting from step 2, we can thus write: $f(n) \in o(f(n))$.
4. By definition, this means that $f(n)$ must be $\leq c \cdot f(n)$ for all constants c for $n \geq$ some constant n_0.
5. If we assume $c = -2$, then our condition in step 4 is contradicted and there exists no n_0 for which $-2 \cdot f(n) > f(n)$ for $n > n_0$. Thus, $o(g(n)) \cap \omega(g(n))$ is the empty set.

2. SPARC to Python

Consider the following SPARC code:

$foo\ x =$
  if $x \leq 1$ **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?

This function takes an integer, and returns 0 or 1 if the integer is 0 or 1 respectively. If integer > 1, then the function recursively repeats the aforementioned process by calling said process on the integer −1 and the integer −2. Then the function recursively sums and returns the # of base case 1's reached.

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?

Work: $O(N)$

Span: $O(N)$

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?

Work: $O(N)$
Span: ~~$O(\log_2 N)$~~ $O(N)$

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?

Work: $O(N)$
Span: $O(\log_2 N)$