CMPS 2200 Assignment 1

Name: _Emily Aymond_

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
  · $2^{n+1} = 2 \cdot 2^n$
  · $2^{n+1} \leq c \cdot 2^n \Rightarrow 2^{n+1} \leq 2 \cdot 2^n$
  · $2^{n+1} \in O(2^n)$

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

  · No
  · $2^{2^n} \leq c \cdot 2^n$
  · no constant c fulfills the inequality
  · $2^{2^n} \notin O(2^n)$

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

  · No
  · $n^{1.01} \leq c \cdot (\log^2 n)$
  · for a large n, $n^{1.01}$ grows faster than $\log^2 n$
  · $n^{1.01} \notin O(\log^2 n)$

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

  · Yes
  · $n^{1.01} \geq \log^2 n$
  · polynomial functions grow larger than polylogarithmic functions, fulfilling the inequality
  · $n^{1.01} \in \Omega \log^2 n$

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

  · No
  · $n^{1/2} \leq c \cdot (\log n)^3$
  · for large n, $n^{1/2}$ grows greater than $(\log n)^3$
  · $\sqrt{n} \notin O((\log n)^3)$

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

  · Yes
  $n^{1/2} \geq c \cdot (\log n)^3$
  polynomial growth dominates logarithmic growth
  $\sqrt{n} \in \Omega((\log n)^3)$

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 . . .

$$foo\ x =$$
```
    if  x ≤ 1  then
        x
    else
        let  (ra, rb) = (foo (x − 1))  ,  (foo (x − 2))  in
            ra + rb
        end.
```

- 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 returns the Fibonacci sequence
· if x is less than or equal to 1, it just returns X
· However, if x is greater than 1, it adds the two
  previous terms, (x-1) and (x-2) to calculate
· the next term in the Fibonacci sequence.
·
·

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 : O(n)
· Span : 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?

  Work: $O(n)$

  Span: $O(\log n)$

- 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?

Work: $O(n \log n)$

Span: $O(n)$