CMPS 2200 Assignment 1

Name: Batu El

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

  $\lim_{n\to\infty} \frac{2^n}{2^{n+1}} = \lim_{n\to\infty} \frac{2^n}{2^n} \cdot \frac{1}{2} = \lim_{n\to\infty} \left(\frac{2}{2}\right)^n \frac{1}{2} = \lim_{n\to\infty} 1^n \cdot \frac{1}{2} = \frac{1}{2} \cdot \lim_{n\to\infty} 1^n = \frac{1}{2} \leq c$ for all $c \geq \frac{1}{2}$.

  So 2^n asymptotically dominates 2^{n+1}, hence $2^{n+1} \in O(2^n)$.

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

  $\lim_{n\to\infty} \frac{2^n}{2^{2^n}} = \lim_{n\to\infty} \frac{2^n}{(2^2)^n} = \lim_{n\to\infty} \left(\frac{2}{4}\right)^n = \lim_{n\to\infty} \left(\frac{1}{2}\right)^n = 0 \leq c$ for all $c \geq 0$.

  Therefore 2^n asymptotically dominates 2^{2^n}, hence $2^{2^n} \in O(2^n)$.

- 1c. Is $n^{1.01} \in O(\log^2 n)$? No

  $\lim_{n\to\infty} \frac{n^{1.01}}{\log^2 n} = \lim_{n\to\infty} \frac{(1.01)\, n^{0.01}}{2 \log n \cdot n^{-1}} = \lim_{n\to\infty} \frac{(1.01)\, n^{1.01}}{2 \log n} = \lim_{n\to\infty} \frac{(1.01)^2\, n^{0.01}}{2\, n^{-1}} = \frac{(1.01)^2}{2} \lim n^{1.01} = \infty > c$ for all c.

  use L'Hôpital since both numerator and denominator goes to ∞

  $n^{1.01} \notin O(\log^2 n)$ since $\log^2 n$ does not asymptotically dominate $n^{1.01}$.

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

  $\lim_{n\to\infty} \frac{\log^2 n}{n^{1.01}} = \lim_{n\to\infty} \frac{2 \log n\, n^{-1}}{(1.01)\, n^{0.01}} = \lim_{n\to\infty} \frac{2 \log n}{(1.01)\, n^{1.01}} = \lim_{n\to\infty} \frac{2\, n^{-1}}{(1.01)^2 n^{0.01}} = \frac{2}{(1.01)^2} \cdot \lim_{n\to\infty} \frac{1}{n^{1.01}} = 0 \leq c$ for all $c \geq 0$.

  use L'Hôpital since both numerator and denominator goes to ∞

  $n^{1.01} \in \Omega(\log^2 n)$ since $n^{1.01}$ asymptotically dominates $\log^2 n$.

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

  $\lim_{n\to\infty} \frac{\sqrt{n}}{(\log n)^3} = \lim_{n\to\infty} \frac{0.5\, n^{-1/2}}{3 \log^2 n \cdot n^{-1}} = \lim_{n\to\infty} \frac{0.5\, n^{1/2}}{3 \log^2 n} = \lim_{n\to\infty} \frac{(0.5)^2\, n^{-1/2}}{(3)(2) \cdot \log n \cdot n^{-1}} = \lim_{n\to\infty} \frac{(0.5)^2\, n^{1/2}}{(3)(2) \log n} = \lim_{n\to\infty} \frac{(0.5)^3\, n^{-1/2}}{(3)(2)\, n^{-1}} = \frac{(0.5)^3}{3.2} \lim_{n\to\infty} \sqrt{n} = \infty > c$ for all c.

  use L'Hôpital since both numerator and denominator goes to ∞

  $\sqrt{n} \notin O((\log n)^3)$ since $\log^3 n$ does not asymptotically dominate $\sqrt{n}$.

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

  $\lim_{n\to\infty} \frac{(\log n)^3}{\sqrt{n}} = \lim_{n\to\infty} \frac{3 \log^2 n \cdot n^{-1}}{(0.5)\, n^{-1/2}} = \lim_{n\to\infty} \frac{3 \log^2 n}{(0.5)\, n^{1/2}} = \lim_{n\to\infty} \frac{3.2 \log n\; n^{-1}}{(0.5)^2\, n^{-1/2}} = \lim_{n\to\infty} \frac{3.2 \cdot \log n}{(0.5)^2\, n^{1/2}} = \lim_{n\to\infty} \frac{3.2 \cdot n^{-1}}{(0.5)^3\, n^{-1/2}} = \frac{3.2}{(0.5)^3} \lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0 \leq c$ for all $c \geq 0$.

  $\sqrt{n} \in \Omega((\log n)^3)$ since $\sqrt{n}$ asymptotically dominates $\log^3 n$

- 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 $10x^2 \notin o(x^2)$.

.

Name		Definition	Intuitively
big-O	:	$O(f) = \{g \in F$ such that f dominates $g\}$	$\leq f$
big-Omega	:	$\Omega(f) = \{g \in F$ such that g dominates $f\}$	$\geq f$
big-Theta	:	$\Theta(f) = O(f) \cap \Omega(f)$	$= f$
little-o	:	$o(f) = O(f) \setminus \Omega(f)$	$< f$
little-omega	:	$\omega(f) = \Omega(f) \setminus O(f)$	$> f$

Prove that $o(g(n)) \cap \omega(g(n))$ is the empty set.

$$o(g(n)) = O(g(n)) \setminus \Omega(g(n)) = O(g(n)) \setminus \big(O(g(n)) \cap \Omega(g(n)) \big)$$

$$\omega(g(n)) = \Omega(g(n)) \setminus O(g(n)) = \Omega(g(n)) \setminus \big(O(g(n)) \cap \Omega(g(n)) \big)$$

$$o(g(n)) \cap \omega(g(n)) = \big[O(g(n)) \setminus (O(g(n)) \cap \Omega(g(n))) \big] \cap \big[\Omega(g(n)) \setminus (O(g(n)) \cap \Omega(g(n))) \big]$$

$$= \{ f(n) \mid f(n) \in O(g(n)) \wedge f(n) \notin \Omega(g(n)) \} \cap \{ f(n) \mid f(n) \in \Omega(g(n)) \wedge f(n) \notin O(g(n)) \}.$$

$$= \{ f(n) \mid [f(n) \in O(g(n)) \wedge f(n) \notin \Omega(g(n))] \wedge [f(n) \in \Omega(g(n)) \wedge f(n) \notin O(g(n))] \} = \emptyset.$$

.
.
.
.
.
.
.
.
.
.
.
.

2. **SPARC to Python**

Consider the following SPARC code:

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

- 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?

.
.
.
.
.
.

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?

.
.
.
.

2.b $foo(0) = 0$

$foo(1) = 1$

$foo(2) = 1$

$foo(3) = foo(2) + foo(1) = (foo(1) + foo(0)) + foo(1) = 1 + 0 + 1 = 2$

$foo(4) = foo(3) + foo(2) = 1 + 2 = 3$

$foo(5) = foo(4) + foo(3) = 3 + 2 = 5$

$foo(6) = foo(5) + foo(4) = 5 + 3 = 8$

$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

foo(x) is a recursive definition of fibonacci sequence, where $f(0) = 0$ and $f(1) = 1$ and for $x > 1$ $f(x)$ is equal to the sum of $f(x-1)$ and $f(x-2)$.

3.b Span $\in O(n)$

Work $\in O(n)$

.
.
.
.
.

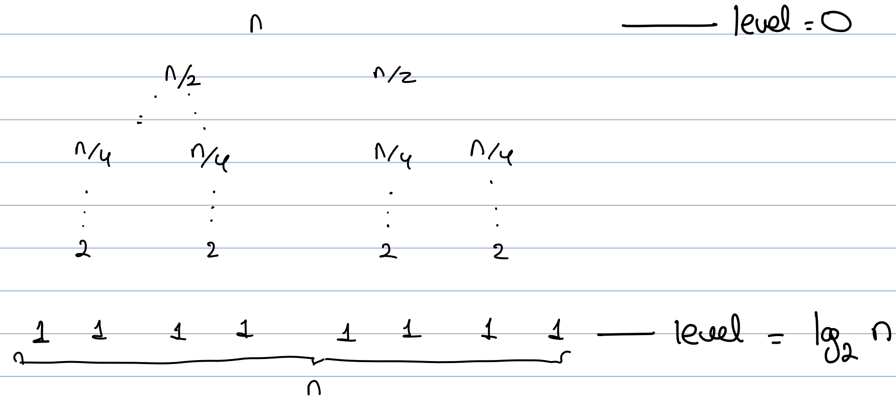
- 3c. Next, implement a `longest_run_recursive`, a recursive, divide and conquer implementation. This is analogous to our implementation of `sum_list_recursive`. To do so, you will need to think about how to combine partial solutions from each recursive call. Make use of the provided class `Result`.
- 3d. What is the Work and Span of this sequential algorithm?

  .
  .
  .
  .
  .
  .
  .
  .
  .
  .
  .

- 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?

.
.
.
.
.
.
.
.

3.d.



Observe that each level contains 2^i nodes

Each node has a unit of computation $O(1)$.

Hence Work = Number of Nodes

Span = Length of the longest path

Height of a perfectly balanced binary tree is $O(\log_2 n)$.

leaf nodes is $2^{\text{height}} = 2^{\log_2 n} = n$

Work $\in O(n)$

Span $\in O(\log n)$.

3.e. Work : $O(n)$

Span : $O(\log_2 n)$

Paralellism : $O\left(\dfrac{n}{\log_2 n}\right)$: exponential speedup.