WEEK 04

1. Preparation for Assignment

If, and *only if* you can truthfully assert the truthfulness of each statement below are you ready to start the assignment.

1.1. **Reading Comprehension Self-Check.**

- I know that *decrease-and-conquer* is a general algorithm design technique based on exploiting a relationship between a solution to a given instance of a problem and a solution to a smaller instance of the same problem.
- I can give a common example of each of the three major variations of the *decrease-and-conquer* technique.
- I know that *insertion-sort*'s notable advantage is good performance on almost-sorted arrays.
- I know that the *topological sorting problem* for a directed graph has a solution if and only if the graph has no directed cycles.
- I know that an application of topological sorting is resolving symbol dependencies in linkers.
- I know that a *dag* is a directed acyclic graph.
- I know that the *decrease-by-one* technique is a natural approach to developing algorithms for generating elementary combinatorial objects.
- I know that *binary search* is the most important and well-known example of a *decrease-by-a-constant-factor* algorithm.
- I know why the Euclidean GCD algorithm is an excellent example of a *decrease-by-a-variable-size* algorithm.
- I know how to order and have ordered the seven Big-$\mathcal{O}$ efficiency classes shown below from *fastest growing* reference function (first) to *slowest growing* reference function (last):
  (1) $\mathcal{O}(\log n)$
  (2) $\mathcal{O}(n!)$
  (3) $\mathcal{O}(n)$
  (4) $\mathcal{O}(1)$
  (5) $\mathcal{O}(2^n)$
  (6) $\mathcal{O}(n \log n)$
  (7) $\mathcal{O}(n^2)$

Date: November 10, 2021.

- I know given a $\mathcal{O}(n^3)$ algorithm whose running time for a problem of size 100,000 is 10 seconds what its running time would be for a problem of size 150,000.
- I know given a $\mathcal{O}(1)$ algorithm whose running time for a problem of size 100,000 is 10 seconds what its running time would be for a problem of size 200,000.

1.2. **Memory Self-Check.**

1.2.1. *Algorithm Efficiency Calculation.*

(1) Code, using Clojure, and understand a *reduce-by-variable-size* algorithm, without looking at pseudocode, that is an implementation of Euclid's GCD algorithm.
(2) Write a non-bruite-force algorithm to generate all subsets of a set (A power set).
(3) Write a non-bruite-force algorithm that solves the Josephus problem.

.

2. Week 04 Exercises

2.1. **Exercise 7 on page 137.**

2.2. **Exercise 1 on page 142.**

2.3. **Exercise 2 on page 148.**

2.4. **Exercise 1 on page 156.**

2.5. **Exercise 1 on page 166.**

2.6. **Exercise 12 on page 167.**

3. Week 04 Problems

3.1. **Exercise 6 on page 137.**

3.2. **Exercise 10 on page 143.** Make sure you write out all the mathematical steps to get the result.

3.3. **Exercise 12 on page 149.**

3.4. **Exercise 8 on page 156.**