WEEK 03

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: October 18, 2018.

- 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 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.** *a.* Since the algorithm uses the formula /gcd(m, n) = gcd(n, m mod n)/, the size of the new pair will be /m mod n/. Hence it can be any integer between 0 and /n - 1/. Thus, the size /n/ can decrease by any number between 1 and /n/.

b. Two consecutive iterations of Euclid's algorithm are performed according to the following formulas:

/gcd(m, n) = gcd(n, r) = gcd(r, n mod r)/ where /r = m mod n/.

Need to show that $n \bmod r \leq n/2$. Consider two cases: $r \leq n/2$ and $n/2 < r < n$. If $r \leq n/2$, then

$n \bmod r < r \leq n/2$.

If $n/2 < r < n$, then

$n \bmod r = n - r \leq n/2$,

too.

2.6. **Exercise 12 on page 167.** +BEGIN$_SRCemacs-lisp : resultssilent(defunpancake-sort(seq)(defunflip(lstindex)(setf(subseqlst0index)(reverse(subseqlst0index))))(loopwithlst = (coerceseq'list)forifrom(lengthlst)downto2forindex = (position(apply'max(subseqlst0i))lst)do(unle$ $index0)(fliplst(1+index)))(fliplsti)finally(return(coercelst(type-ofseq)))))+ END_SRC(source : https : //rosettacode.org/wiki/Sorting_algorithms/Pancake_sort)+ RESULTS :: pancake - sort$

+BEGIN$_SRCemacs - lisp(pancake - sort'(678925341)) + END_SRC$

+RESULTS: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

- Place spatula under largest pancake. - Flip the stack over (the largest pancake will now be on top). - Place the spatula under the bottom of the stack and flip over the whole stack again (the largest pancake will now be on the bottom). - After flipping over the whole stack, increment a counter (/flips/). - Search for the next biggest pancake. Reduce the search to /number of pancakes/ minus the number of times the stack has been flipped to place the next largest pancake on the bottom (/flips/), starting from the top of the stack. - After finding the next biggest pancake, place the spatula under the /flips+1/ pancake. Flip the stack over and continue.

*** Answer 1. If /f/ is equal to /n/, stop sorting. Look for the largest value starting from the top (/n/) until you reach the value located at position /f/ (number of flips). Stop looking when the largest value between /f/ and /n/ is found. 2. Place the flipper under the chosen value. 3. Flip over the stack of values above the flipper. 4. Place the flipper under the value located at position /f+1/ (indexing from the bottom). 5. Flip over the stack above the flipper. 6. Increment /f/ by 1. 7. Repeat until the exit condition is met.

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.**