WEEK 08

1. Preparation for Assignment

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

1.1. Reading Comprehension Self-Check.

- I know why it is **false** to say that **dynamic programming** is better than static programming (such as is taught in CS124 (or equivalent)).
- I know why it is **false** to say that applicability of dynamic programming to an optimization problem requires the problem to satisfy the *principle of suboptimality*: a suboptimal solution to any of its instances must be made up of suboptimal solutions to its subinstances.
- I know how to discover that one of the earliest applications of dynamic programming is an algorithm that solves the traveling salesman problem in time $\mathcal{O}(n^2 2^n)$.
- I know why it is **false** to say that solving a **continuous** knapsack problem by a dynamic programming algorithm exemplifies an application of this technique to difficult problems of combinatorial optimization.
- I know how *memory functions* such as this Fibonacci C++ code constitute a space-grabbing but time-saving technique.
- I know how *memory functions* such as this Elisp Knapsack code constitute a space-grabbing but time-saving technique.
- I know how many *distinct* binary search trees can be constructed for a set of 4 orderable keys: A, B, C and D.
- I know how to draw all optimal BSTs for a set of 4 orderable keys.
- I know how to label each tree I drew with its unique level-order traversal (like ABCD).
- I know how to draw the optimal BST given 0.1, 0.2, 0.3, and 0.4 as the probabilities of the four keys A, B, C, and D.
- I know how to compute the average search cost for an optimal BST.

1.2. Memory Self-Check.

Compare and contrast **dynamic programming** and **divide and conquer** by putting an X in a table cell if the property is true:

Date: November 12, 2018.

Property	Dynamic Programming	Divide and Conquer
Works bottom up		
Works top down		
Divides problems into subproblems		
Subproblems may be overlapping		

(*Bottom up* means starting at smallest or simplest subproblem, *then* combining subproblem solutions of increasing size until the solution of the original problem is reached.)

2. Week 08 Exercises

2.1. **Exercise 4 on page 290.**

2.2. **Exercise 1 on page 296.**

2.3. **Exercise 2 on page 303.**

2.4. **Exercise 5 on page 303.**

2.5. **Exercise 2 on page 311.**

3. Week 08 Problems

3.1. **Not in the Book.** In a language of your choice, implement the search elements of an optimal binary search tree for a set of n keys. Test the time efficiency of your code for the following cases:

(a) Search for the largest element.
(b) Search for the smallest element.
(c) Search for an element not present in the tree.