Homework 2: Backtracking and Dynamic Programming
DUE: 11:30pm Mon, Mar 1, 2021, on Blackboard

- Please type and submit PDF file to Blackboard. No late submissions will be accepted.
- **Academic honesty**: at the beginning of your paper, please include: "I understand and agree to abide by the provisions in the University of Houston Undergraduate Academic Honesty Policy" and "This is my own work."
- Discussion is encouraged. In each problem solution, list the names of persons that you have discussed with about that problem, except for large group discussion such as in MS Teams where a note of "group discussion" suffices. If an online source helped you, list the URL.
- **Unless otherwise statement, for an algorithm design problem, please include algorithm description (texts + pseudocode), proof of correctness, and analysis of worst case time complexity (big O or big Theta)**.

Backtracking:

P0 (15pt): Give an algorithm to print out all possible subsets of an n-element set, such as $\{1, 2, 3 \ldots, n\}$.

P1 (15pt): A *derangement* is a permutation p of $\{1, \ldots, n\}$ such that no item is in its proper position; i.e. $p_i \neq i$ for all $1 \leqslant i \leqslant n$. For example, $p = \{3, 1, 2\}$ is a derangement of {1,2,3}, whereas $p = \{3, 2, 1\}$ is not.

Write an efficient backtracking program that prints out all derangements of $\{1, \ldots, n\}$.

P2 (20pt): Describe recursive algorithms for the following generalizations of the SubsetSum problem:

(a) Given an array $X[1..n]$ of positive integers and an integer T , compute the number of subsets of X whose elements sum to T .

(b) Given two arrays $X[1..n]$ and $W[1..n]$ of positive integers and an integer T, where each $W[i]$ denotes the weight of the corresponding element $X[i]$, compute the maximum weight subset of X whose elements sum to T . If no subset of X sums to T , your algorithm should return $-\infty$.

Dynamic Programming:

P3 (20pt): In a strange country, the currency is available in the following denominations: \$1, \$4, \$7, \$13, \$28, \$52, \$91, \$365. Find the minimum bills that add up to a given sum \$$k$.

(a) The greedy change algorithm repeatedly takes the largest bill that does not exceed the target amount. For example, to make \$122 using the greedy algorithm, we first take a \$91 bill, then a \$28 bill, and finally three \$1 bills. Give an example where this greedy algorithm uses more bills than the minimum possible. [Hint: It may be easier to write a small program than to work this out by hand.]

(b) Describe and analyze a recursive algorithm that computes, given an integer k, the minimum number of bills needed to make \$$k$. (Don't worry about making your algorithm fast; just make sure it's correct.)

(c) Describe a dynamic programming algorithm that computes, given an integer k, the minimum number of bills needed to make \$$k$. (This one needs to be fast.)

P4 (30pt): Eggs break when dropped from great enough height. Specifically, there must be a floor f in any sufficiently tall building such that an egg dropped from the fth floor breaks, but one dropped from the $(f-1)$st floor will not. If the egg always breaks, then $f=1$. If the egg never breaks, then $f=n+1$.

You seek to find the critical floor f using an n-story building. The only operation you can perform is to drop an egg off some floor and see what happens. You start out with k eggs, and seek to drop eggs as few times as possible. Broken eggs cannot be reused. Let $E(k,n)$ be the minimum number of egg droppings that will always suffice.

1. (a) Show that $E(1,n)=n$.
2. (b) Show that $E(k,n)=\Theta(n^{1/k})$. [Hint1: Try to recurse; Hint2:mathematical induction on k. Hint3: treat variables as real numbers and minimize via derivative.]
3. (c) Find a recurrence for $E(k,n)$. What is the running time of the dynamic pro- gram to find $E(k,n)$? [Note: this is different from (b). k,n, $E(k,n)$ have to be integers.]