EE360C: Algorithms
University of Texas at Austin — Homework #7
Dr. David Soloveichik — Due: November 15, 2018 (*in-class quiz*)

Homework #7

You do not need to turn in these problems. The goal is to reinforce what we learned in class, as well as to cover material we didn't have time to cover in class. The in-class quiz that will cover the same or similar problems. Material on homework can also appear on exams.

Problem 1: Knapsack: which items to take?

In class we developed the dynamic programming solution for the knapsack problem, which allows us to compute the *value* of the optimal solution (OPT) in $\Theta(nW)$ time. In this problem you will extend this solution to compute which items the thief should take.

(a) At first glance we might think that we can compute the optimal solution at the same time as we are computing the optimal value. Consider the following naive modification (underlined code) of the iterative knapsack solution we saw from class, and explain why it does not work. Hint: why can't we start printing items to take until we compute all the way up to $M[n, W]$?

```
input: item weights $w_1,…,w_n$ and values $v_1,…,v_n$ and knapsack capacity W

for w = 0 to W
  M[0, w] = 0

for i = 1 to n
  for w = 1 to W
    if ($w_i$ > w)
       M[i, w] = M[i–1, w]
     else
       M[i, w] = max {M[i–1, w],  $v_i$ + M[i–1, w–$w_i$]}
       if M[i–1, w] < $v_i$ + M[i–1, w–$w_i$]
          print i              //item i is in optimal solution

return M[n, W]
```

(b) Describe how once M is computed, we can generate the list of items that the thief should take. What is your algorithm's asymptotic running time?

Problem 2: Summing Integers

Suppose you are given a collection $A = \{a_1, a_2, \ldots, a_n\}$ of n positive integers that add up to $2Z$. Design an $O(nZ)$ time algorithm to decide if the set can be partitioned into two groups B and $A - B$ such that:

$$\sum_{a_i \in B} a_j = \sum_{a_i \in (A-B)} a_i = Z$$

Problem 3: Dynamic Programming for Change

You are given k denominations of coins, $d_1, d_2, \ldots d_k$ (all integers). Assume $d_1 = 1$ so it is always possible to make change for any amount of money. Give a dynamic programming algorithm that makes change for an amount of money n using as few coins as possible. What is its running time? (Recall from a past homework that the natural greedy algorithm doesn't work for all sets of coin denominations. The dynamic programming algorithm should always work.)

Problem 4: End of Semester Dynamic Programming

It's the end of the semester, and you're taking n courses, each with a final project. Each project will be graded on a scale of 1 to $g > 1$, where a higher number is a better grade. Your goal is to maximize your average grade on the n projects. (Note: this is equivalent to maximizing the *total* of all grades, since the difference between the total and the average is just the factor n.)

You have a finite number of hours $H > n$ to complete all of your course projects; you need to decide how to divide your time. H is a positive integer, and you can spend an integer number of hours on each project. Assume your grades are deterministically based on time spent; you have a set of functions $\{f_i : 1, 2, \ldots, n\}$ for your n courses; if you spend $h \leq H$ hours on the project for course i, you will get a grade of $f_i(h)$. The functions f_i are nondecreasing; spending more time on a course project will not *lower* your grade in the course.

To help get you started, think about the (i, h) subproblem. Let the (i, h) subproblem be the problem that maximizes your grade on the first i courses in at most h hours. Clearly, the complete solution is the solution to the (n, H) subproblem. Start by defining the values of the $(0, h)$ subproblems for all h and the $(i, 0)$ subproblems for all i (the latter corresponds to the grade you will get on a course project if you spend *no* time on it).

Give the dynamic programming equation to define the value of the optimal solution (i.e., the value of any (i, h) subproblem, where $0 \leq i \leq n$ and $0 \leq h \leq H$), describe an algorithm for iteratively computing the value of the optimal solution, and describe an algorithm for recreating the optimal solution (i.e., mapping your H hours to your n projects).