Problem Set 3—Due Wed. May. 9, 3:15PM

Write clearly (if you can't, type it). Illegible answers won't get points. Be sure to give a high level overview in addition to more details (which could, but need not always, be pseudo code). Also, when giving an algorithm, we always want an efficient algorithm, and you should justify its correctness and run time.

(13) Problem 1. Suppose we have n boxes we want to stack. The ith box has a an integer weight w_i and a strength s_i. The total weight of the boxes on top of box i plus w_i can be at most s_i (so a box's own weight limits what can be on top of it). Our goal is to find the best sequence of boxes to stack such that the total weight of the boxes used is as large as possible, and the total weight on top of each box is at most that box's strength.

For example, with boxes (3,3), (10,15), (8,30), (3,25) (16,25) We could stack (left most = bottom of the stack):

(16,25), (8,30) For a total weight of 24 (note we can't add any box since then we would have more than 25 on the bottom box)

A better solution is: (8,30) (16,25) (3,3) for a total of 27 (note we can't use (10,15) instead of (3,3) since then we would have too much weight for the second box in our stack).

a) Prove that there is an optimal solution in which the strengths of the boxes decrease (or are tied) as we go up the stack.

b) Suppose all boxes have the same strength S. Describe how to use dynamic programming to find an optimal solution.

c) Now consider the general case where weights and strengths may vary. Describe how to use dynamic programming to find an optimal solution. Hints: i) first sort the boxes by strength, ii) consider optimal solutions using the first i strongest boxes, for each i, iii) compare to the knapsack solution.

(13) Problem 2. Rod-Cutting variations:

a)Consider a variant of the rod-cutting problem where we can only sell a piece of size i once (thus once we cut, e.g. a piece of size 3, there is no further demand for rods of size 3, so we don't want to cut any more of that size). Note that in this setting, the best solution might wind up with a final piece of rod we can't sell. Give an efficient dynamic programming solution to this problem and analyze its time and space usage (hint, you may find the solution to another problem useful in solving this).

b) Now consider a different setting: as we cut the rod, it gets weaker, so we can make at most $k < n$ cuts. Thus, now k, n, and the price array P, are inputs to the problem. Give an efficient dynamic programming solution to this problem and analyze its time and space usage. Note that as in the original setting (discussed in class and in the book) we can cut multiple rods of the same size and sell them.

(35) Problem 3. *Activity Varients*

Suppose that we have n activities to schedule and as in the class example each activity has a start time s_i and a finish time f_i, and we cannot schedule two activities if they overlap. As a new variant, each activity has a weight w_i which is the profit you get for scheduling activity i. Our goal is to find a set S of non-overlapping activities which have maximum total weight. You may assume all the weights are distinct.

Part A. Consider the following greedy strategy:

(1) sort the activities so $w_1 > w_2 > \ldots > w_n$.

```
(2)  For i = 1 to n
       Put activity i into S unless it overlaps an activity already in S.
```

Give an example that shows that this strategy will **not** always find an optimal solution.

Part B.

Suppose now (but not for parts C,D below) that we have many ties among the weights, so there are only k distinct weights and k is much smaller than n (the number of activities). Rather than sorting the items as is done in (1) above, we could instead first find the k distinct weights among the items and then sort them.

Describe a fast way to find the k distinct weights and give the running time of your implementation of the algorithm of part A).

Part C. Now suppose each activity has length 1 (so $f_i = s_i + 1$) and the start (and thus end times) are integers. Prove that the algorithm of part A) finds an optimal solution.

Part D. For the setting of part C (length one, and weights), the algorithm of part A would take $\Theta(n \log n)$ time for sorting. Instead, desribe how to use hashing to find an optimal solution for this setting in $O(n)$ expected time.

(25) Problem 4. We know that the 0-1 knapsack problem can be solved by Dynamic programming, and the fractional problem can be solved more quickly by the greedy algorithm. In many settings there is a mixture of items: some can be split (as in the fractional problem) while others cannot. Suppose we have n items which can be split and k items which cannot. Let $(v_1, w_1), \ldots (v_n, w_n)$ be the value weight pairs for the fractional items, and $(v_{n+1}, w_{n+1}) \ldots (v_{n+k}, w_{n+k})$ be the 0-1 items. As before we have a single weight limit W.

Part A. Suppose that $k = 1$. Describe a fast algorithm to find an optimal choice of items. Give the running time of your solution and justify its correctness.

Part B. Describe how to solve this problem in general, but you may assume that k is much smaller than n.