Big Bucks
Smallville is the last city on Earth not saturated by Big Bucks coffee shops. Smallville has one business street with $n$ blocks. The profit associated with putting a coffee shop on block $i$ in given in an array $Profit[i]$. However, they cannot put coffee shops within $d \geq 1$ blocks from each other, i.e., if a shop is in block $i$ then there cannot be one in blocks $i - d, i - d + 1, i - 1$ or $i + 1, i + 2, ... i + d$.

An backtracking algorithm for computing the maximum total profit of BigBucks coffee shops is as follows:

$$BTBigBucks(d, Profit[1..n])$$

1. IF $n = 0$ return 0.
2. IF $n \leq d + 1$ return $\max_{1 \leq I \leq n} Profit[I]$
3. Case1 $\leftarrow Profit[1] + BTBigBucks(d, Profit[d+2..n])$ \{If we put a shop in block 1, we cannot put one in $2...d+1$\}
4. Case2 $\leftarrow BTBigBucks(d, Profit[2..n])$ \{If we don’t put a shop in block 1, there are no other restrictions\}
5. Return $\max(\text{Case1}, \text{Case2})$

Part 1: 5 points Illustrate this algorithm on the following inputs: $d = 2, n = 8, Profit[1..8] = 2, 4, 3, 7, 8, 4, 7, 5$ (as a tree of recursive calls and answers).

Part 2: 5 points Give an upper bound on the number of recursive calls the above algorithm makes, in the worst-case. (Some points will be based on how tight the bound is. Be sure to explain your answer.)

Part 3: 10 points Give a dynamic programming version of the recurrence.

Part 4: 5 points Give a time analysis of this dynamic programming algorithm.

Part 5: 5 points Show the array that your dynamic programming algorithm produces on the above example.

For each of the following four problems, describe the fastest dynamic programming algorithm you can find, and give a time analysis (in terms on any of the given parameters).

One Dimensional Clustering: 20 pts You are given $n$ real numbers $r_1, r_2, ..., r_n$ and an integer $1 \leq k \leq n$. You want to find $k$ disjoint intervals $I_1$ =
Given an efficient algorithm for this problem. Our best time is $O(n^2k)$.

**Descending partitions-20pts** A descending partition of positive integer $N$ is a sequence of positive integers $A_1 > A_2 > ... > A_k$ with $\sum_{i=1}^{k} A_i = N$. Give an efficient (poly-time in $N$) algorithm that, given $N$, computes the NUMBER of decreasing partitions of $N$. For example, if $N=6$, the decreasing partitions are: $(6); (5,1); (4,2); (3,2,1)$ so your algorithm, on input 6 should return 4. (14 points correct poly-time algorithm, 6 pts. efficiency, e.g. $N^2$ vs. $N^3$ time)

**Library storage-20pts** A library has $n$ books that must be stored in alphabetical order on adjustable height shelves. Each book has a height and a thickness. The width of the shelf is fixed at $W$, and the sum of the thicknesses of books on a single shelf must be at most $W$. The next shelf will be placed on top, at a height equal to the maximum height of a book in the shelf. You are given the list of books in alphabetical order, $b_i = (h_i, t_i)$, where $h_i$ is the height and $t_i$ is the thickness, and must organize the books in that order.

**Weighted Oasis Problem: 20 pts** Consider the following weighted version of the Oasis problem from last class. You are planning a trip through the desert, that must follow a fixed route that passes through several oases. You have as input an list of $n$ Oases, each oasis $O_i$ with a distance $d_i$ from the start, and a price $p_i$ of staying there. You can assume $O_i$’s are listed in order they will be encountered, so in increasing order of $d_i$. $O_n$ is your destination. There is also an input $M$, the maximum distance you can travel between stops. You need to pick a set of oases $S$, so that the first oasis in $S$ is distance at most $M$ from the start, the distance between consecutive pairs is at most $M$, and the destination $O_n$ is in $S$. Given these constraints, you wish to minimize your total expenses, $Cost(S) = \sum_{O_i \in S} p_i$. (Our best time is $O(n^2)$.)