Gizmos (20 points) Consider the following problem. You wish to purchase (at least) \( n \) identical gizmos. Gizmos come in packages of different sizes and different prices. You can buy any number of packages of each size, as long as the total number is at least \( n \). You wish to find the minimum total price of such a set of packages.

The input is given as \( n \) and an array \( \text{Packages}[1..m] \), where each \( \text{Package}[i] \) has a positive integer field \( \text{Package}[i].\text{size} \) and a positive real field \( \text{Package}[i].\text{price} \) giving the number of gizmos in the package and the price of the package.

A recursive algorithm to solve this problem is:

\[
\text{BestPrice}[n : \text{positive}\text{integer}, \text{Packages}[1..m] : \text{array of pairs (size: integer, price: real)}] \\
1. \text{MinPrice} \leftarrow \infty; \\
2. \text{For } d = 1 \text{ to } m \text{ do:} \\
3. \hspace{1em} \begin{align*}
4. & \text{IF } \text{Packages}[d].\text{size} \geq n \text{ THEN } \text{TempPrice} \leftarrow \text{Packages}[d].\text{price} \\
5. & \hspace{1em} \text{ELSE } \text{TempPrice} \leftarrow \text{Packages}[d].\text{price} + \\
6. & \hspace{2.5em} \text{BestPrice}(n - \text{Packages}[d].\text{size}, \text{Packages}); \\
7. & \text{IF } \text{TempPrice} < \text{MinPrice} \text{ THEN } \text{MinPrice} \leftarrow \text{TempPrice}; \\
8. \end{align*} \\
9. \text{end;} \\
10. \text{Return MinPrice.}
\]

Part 1: 2 points Show the recursion tree of the above algorithm on the following input: \( n = 6 \), packages: buy 5 for $12, 3 for $8 or 2 for $6.

Part 2: 3 points Give a bound on the worst-case number of recursive calls the recursive algorithm could make in terms of \( n \) and \( m \).

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

Part 4: 3 points Give a time analysis of this dynamic programming algorithm, in terms of \( n \) and \( m \).

Part 5: 2 points Show the array that your algorithm produces on the above example.

For each of the following three problems, describe the fastest dynamic programming algorithm you can find, and give a time analysis (in terms on any of the given parameters).
Descending partitions-20pts A descending partition of positive integer $N$ is a sequence of positive integers $A_1 > A_2 > \ldots > A_k$ 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. (7 points correct poly-time algorithm, 3 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. Give an algorithm that minimizes the total height of shelves used to store all the books. 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 Single Room Conference Schedule -20pts In class, we saw a greedy algorithm for choosing the most non-overlapping events that could be scheduled in a conference room. In this version of the problem, every event has a weight, that gives its importance, and the objective is to maximize the total weight, not the total number. Formally

Instance: A set of $n$ events $E_i = < s_i, f_i, w_i >$, $1 \leq i \leq n$, where $s_i < f_i$ and $0 < w_i$. $s_i$ is called the start time of the event, $f_i$, the finish time, and $w_i$ the weight.

Solution Format: A subset $S$ of the events.

Constraints: $S$ cannot contain two events $E_i$ and $E_k$, $i \neq k$, that overlap, i.e., where $s_i \leq s_k \leq f_i$.

Problem: Find a legal set of events $S$ which has the maximum possible total weight, $(\text{Maximize } W(S) = \sum_{j \in S} w_j)$.

Hint: It might help to sort the events by their start times. Start with a recursion which considers whether or not the first (earliest to start) job should be in $S$.

Implementation (20 pts) Implement the Descending Partitions algorithm above. Give the number of partitions for all numbers in the range $n = 100$ to $n = 120$. 

2