## Dynamic Programming III: Knapsack

#### The Problem

Imagine that you are on a hike and you find a cave filled with riches: n riches to be exact.

Each item i in the cave has some value  $v_i$ . But it also has a weight  $w_i$  to it, and your hiking pack (or knapsack) can only hold up to a total weight W.

Our goal is to pick which items S to take to maximize the value  $\sum_{i \in S} v_i$  in your pack, but ensure that the weight doesn't exceed the maximum allotted,  $\sum_{i \in S} w_i \leq W$ .

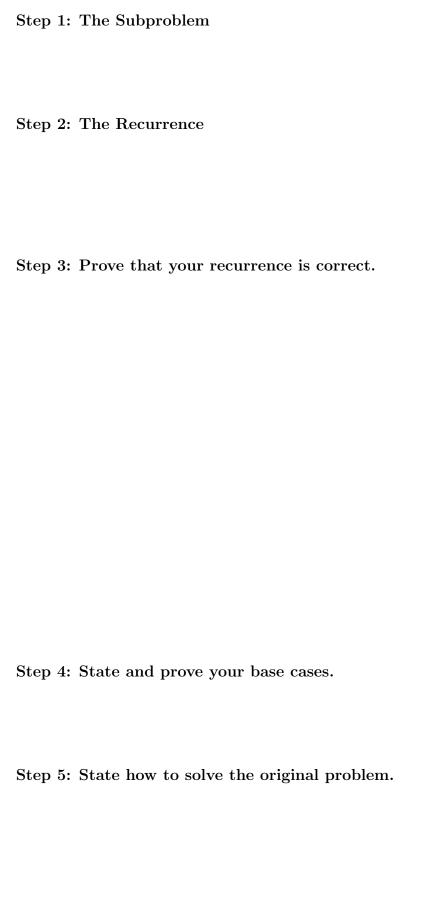
### Making the Key Observation

What key observation can we make that will help us move toward our subproblem and recurrence? A reminder of our other key observations:

Scheduling: Either the last job is in the solution, or it isn't. (Then what does that mean for the rest of the schedule and the maximum weight?)

Least segmented squares: The last point  $p_n$  belongs to a single segment which must begin somewhere. Where does it begin? In each case, what does the minimal error and optimal solution look like?

#### Now for knapsack:



# Step 6: The Algorithm

## Step 7: Running Time

- a. Pre-processing: computing base cases, sorting, etc.
- **b.** Filling in memo: This can be further broken down into
  - (a) Number of entries of your memo table.
  - (b) Time to fill each entry. Be careful of things like taking maxes over n elements!
- c. Postprocessing: Return statement, etc.

**Definition 1.** An algorithm is *pseudopolynomial* time if its runtime is polynomial in the numerical values of the inputs, but not the number of bits required to express them.