Problem 1
You just bought an $m$ gallon (i.e. $128m$ ounce) leather bag and want to fill it with spices. You are at your friend’s spice shop, and all spices for sale are in powder form. There are $n$ different spices available, and there are $s_i$ ounces of spice $i$ available for each $i = 1, \ldots, n$. Each $s_i$ is a positive integer. Furthermore, you know that the market price for spice $i$ is $p_i$ dollars per ounce. Your friend, being the great friend that she is, is willing to let you fill your 128$m$ gallon bag with as much of her spices as you wish, for free! Naturally, you decide to pack the most value you can into your bag that fits. Spices, being in powdered form, can be packed into your bag in any fractional amount. Give an algorithm to decide which spices, and how much of each spice, to pack in your bag. You may assume addition, subtraction, multiplication, and comparison are constant time operations. Note: full points will be given for running time $O(n \log n)$, though a bonus point will be given for running time $O(n)$.

Problem 2
You are (yet again) given $n$ boxes labeled 1, 2, $\ldots$, $n$. For each box you are given its “strength” $s_i$, which means that it can support up to $s_i$ pounds of weight above it without it collapsing. Each box weighs one pound.

You would like to group the given boxes into $M$ stacks such that no box has more weight above it in its stack than its strength, lest some box collapse. Your goal is to do this with as small a number $M$ of stacks as possible.

(a) (3 points) Show that there is always an optimal solution for which in each stack, the boxes are sorted by strength where the topmost boxes in the stack have the lowest strength.
(b) (7 points) One of your CS 124 TF’s has implemented the following greedy algorithm. The algorithm processes the boxes from weakest to strongest. Initially it only maintains one stack, which is empty. When box $i$ is processed, the algorithm inserts it as the bottom-most box in one of the current stacks which it can support (i.e. which has size at most $s_i$). If there are multiple stacks it can support, it picks one via some tie-breaking mechanism that we do not understand. Otherwise, if box $i$ cannot support any stack, the algorithm allocates a new stack with $i$ as its sole box. Prove that this algorithm produces an optimal solution. You do not need to analyze runtime. **Hint:** use induction.

Problem 3

Figure 1: Example input to Problem 3.

You are given a rooted tree on $n$ nodes as input, where the root is vertex 1. The input is specified as follows: you are given an array $A[1..n]$ such that $A[i]$ is a linked list containing the children of vertex $i$. Each vertex in this tree corresponds to a computational task that must be carried out by one of your servers. However, there is a constraint that a vertex cannot be processed by a server until all its children have already been processed. You have $k$ servers available to do the processing, meaning that you can only process up to $k$ tasks in parallel. Every task takes one unit of time to complete. Your goal is to devise a greedy algorithm to compute the optimal time needed to complete the root task of the tree.

For example in Figure 1 with $k = 2$, a valid ordering is to process vertices 2 and 5, then 6 and 7, then 8 and 9, then 3 and 4, then 1, taking 5 units of time. This is optimal.

(a) (4 points) Show that if the number of leaves is at most $k$, then the minimum time needed to complete all tasks is equal to the height of the tree plus 1 and give a greedy algorithm to calculate an optimal scheduling of the tasks. **Note:** we interpret a tree with a single node as having height 0.

(b) (3 points) Consider the general case, where the number of leaves may be larger than $k$. Let $H$ be the height of the tree, and show that for any $0 \leq h \leq H$, $h + 1 + \lceil \frac{n_h}{k} \rceil$ is a lower bound on the processing time for any schedule of processing tasks. Here $n_h$
denotes the number of nodes at depth \( h + 1 \) or deeper in the tree, where the “depth” of a node is its distance to the root.

(c) (3 points) Devise a greedy algorithm to find an optimal scheduling of tasks even in the general case where the number of leaves may be larger than \( k \). In your correctness proof, you may wish to show that your algorithm achieves processing time matching the lower bound from (b) for some \( h \).

**Programming Problem (EXTRA CREDIT)**

Solve **OLDMAN** on the programming server (https://cs124.seas.harvard.edu).

**Hint:** use divide-and-conquer (an approach covered in lecture on March 1st). Though the non extra-credit portion of this problem set is due March 7th, you have until the **Problem Set 5 deadline** to complete this extra credit problem (if you choose to do it).

Because this is extra credit, there will be **no collaboration and no staff help** with this problem. Extra credit will be considered at the end of the semester.