Fundamental Algorithms, Section 003
Homework 8, Fall 22. Due date: Wednesday, November 09, 4:55pm on Gradescope

1. In the knapsack problem the input consists of n items of sizes $s_1, s_2, \ldots, s_n$ and values $v_1, v_2, \ldots, v_n$, respectively. There is also an input parameter t. The task is to find a subset of the items of maximum value with total size at most t. All the parameters are positive integers.

Give pseudo-code for an efficient recursive implementation of the following recursively defined solution to the the knapsack problem.

$$V(n,t) = \begin{cases} V(n-1,t) & \text{if } s_n > t \text{ and } n \geq 1 \\ \max\{V(n-1,t), V(n-1,t-s_n)+v_n\} & \text{if } s_n \leq t \text{ and } n \geq 1 \end{cases}$$
$$V(0,t) = 0$$

Note that you are being asked for a recursive implementation, not an iterative implementation. What is the running time of your implementation? Justify your answer.

2. Consider the problem of constructing an optimal binary search tree (BST) T. Suppose T is storing the items $e_1 < e_2 < \ldots < e_n$, and item e_i will be accessed s_i times. Our cost measure is the number of comparisons performed to access e_i (where a comparison is now a 3-way decision: $>$, $=$, or $<$). Equivalently, it is the number of nodes on the path from the root of T to e_i inclusive.

The cost of an optimal BST is given by the following recurrence.

$$D(i,k) = \min_{i \leq j \leq k} \{D(i,j-1) + D(j+1,k)\} + \sum_{j=i}^{k} s_j$$
$$D(i,i-1) = 0$$

The following code provides an efficient recursive solution. One small gain in efficiency stems from pre-computing $\text{Tot}[j] = \sum_{i=1}^{j} s_i$ for $0 \leq j \leq n$, for then $\sum_{j=i}^{k} s_j$ is given by $\text{Tot}[k] - \text{Tot}[i-1]$, an $O(1)$ time calculation.

```
OptBST($i,k$)
if $i = k+1$ then Cost[$i,k$] $\leftarrow$ 0
else
    Cost[$i,k$] $\leftarrow$ MaxInt;
    RootCost $\leftarrow$ Tot[$k$] $-$ Tot[$i-1$];
    for $j = i$ to $k$ do
        if not(Done[$i,j-1$]) then OptBST($i,j-1$)
        end if
        if not(Done[$j+1,k$]) then OptBST($j+1,k$)
        end if
        Cost[$i,k$] $\leftarrow$ min{Cost[$i,k$], Cost[$i,j-1$] + Cost[$j+1,k$] + RootCost};
    end for
end if
Done[$i,k$] $\leftarrow$ True
```

The driver initializes all entries in Done to FALSE, sets Tot[0] = 0, computes $\text{Tot}[j] = \text{Tot}[j-1] + s_j$ for $1 \le j \le n$, and then calls OptBST$(1, n)$. Note that the range of the first argument in Done is $[1 : n]$ and the range for the second argument is $[0 : n]$.

Your task: Show how to augment this code so as to recover an optimal binary search tree, and give pseudo-code to produce the BST T. You may assume you have a procedure GetNode(v) which returns a node with left and right pointers plus a .item field to hold the item being stored at that node.

What is the running time of this algorithm? Please give separate bounds for the augmented dynamic program and for the subsequent procedure to build the optimal BST. Justify your answers.

3. Suppose your monetary system has n types of coin of value $v_1, v_2, \ldots, v_n$, respectively, where each v_i is an integer. Suppose that $v_1 < v_2 < \ldots < v_n$. You are to give an algorithm to determine whether one can select k or fewer coins, possibily with some or all of them having the same value, so that their total value is v.

a. Give a recursive formulation of a solution to this problem.

b. What is the running time of an efficient recursive implementation? A solution that is polynomial in n, k and v is the goal. Justify your answer.

c. What additional information would you need to compute the values of the at most k coins whose total value is v, if this is possible?

4.a. Consider the problem of merging a sequence of sorted lists $L_1, L_2, \ldots, L_n$ of lengths $\ell_1, \ell_2, \ldots, \ell_n$. In this problem we are only allowed to merge adjacent lists: the allowed action is to remove two adjacent lists from the sequence and then replace them in the sequence with their merge. Thus, if after some sequence of merges we have obtained the shorter sequence $L'_1, L'_2, \ldots, L'_m$, and we merge L'_i and L'_{i+1} yielding L''_i, then the new sequence is $L'_1, \ldots L'_{i-1}, L''_i, L'_{i+2}, \ldots, L'_m$.

Suppose it costs $|L_a| + |L_b|$ to merge lists L_a and L_b, for any pair of lists L_a and L_b.

a. Give a recursive formulation of the cost of an optimal merging sequence.
Hint. Consider the possible choices for the final merge.

b. What is the running time of an efficient recursive implementation? Justify your answer.

c. What additional information would you need to compute the tree representing an optimal merge sequence?

Challenge problem. Do not submit.