Homework 8, Solution set

1. Val$[i, j]$ is an array that stores the value for a knapsack of size j for the first i items, with $0 \le i \le n$ and $0 \le j \le t$. It is initialized to be all zero; similarly, the array Done$[i, j]$ is initialized to False for the same range of i, j values. Following this, OptKnap(n, t) is called.

OptKnap(n, t)
if $n = 0$ or $t = 0$ **then** Val$[n, t] \leftarrow 0$
else
  if **not**(Done$[n-1, t]$) **then** OptKnap$(n-1, t)$
  end if
  if **not**(Done$[n-1, t-s_n]$) and $t \ge s_n$ **then** OptKnap$(n-1, t-s_n)$
  end if
  Val$[n, t] \leftarrow \max\{$Val$[n-1, t]$, Val$[n-1, t-s_n] + v_n\}$;
end if
Done$[n, t] \leftarrow$ True

There are up to $(n+1) \cdot (t+1)$ recursive calls that might be made. Each recursive call has an $O(1)$ cost for its non-recursive computation. Thus the overall cost is $O(nt)$.

2. We introduce a new array Root$[i, k]$ with $1 \le i \le n$ and $1 \le k \le n$; it will store the index of the best choice for the item to go at the root of the BST for items $e_i, e_{i+1}, \ldots, e_k$.

The following is the enhanced recursive procedure.

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
    if Cost$[i, k] >$ Cost$[i, j-1]$ + Cost$[j+1, k]$ + RootCost **then**
      Root$[i, k] \leftarrow j$; **(* this is the code enhancement *)**
      Cost$[i, k] \leftarrow$ Cost$[i, j-1]$ + Cost$[j+1, k]$ + RootCost
    end if
  end for
end if
Done$[i, k] \leftarrow$ True

After the run of OptBST$(1, n)$ we can build an optimal BST with a call to BuildOptBST$(1, n, T)$, which returns T, or more precisely, a pointer to its root.

BuildOptBST(i, k, v) (* returns v, a pointer to the root of the tree *)
if $i > k$ **then** $v \leftarrow$ nil
else
GetNode(v);
$j \leftarrow$ Root$[i : k]$;
$v.$item $\leftarrow e_j$;
BuildOptBST$(i, j-1, w)$;
BuildOptBST$(j+1, k, x)$;
$v.$left $\leftarrow w$;
$v.$right $\leftarrow x$
end if

The procedure call OptBST$(1, n)$ makes up to $n(n+1)$ recursive calls (including the initial call). The call OptBST(i, k) has non-recursive cost $O(k-i+1) = O(n)$. Thus the overall cost is $O(n^3)$ (and actually is $\Theta(n^3)$, though we have not shown that).

The procedure BuildOptBST$(1, n, T)$ runs in $O(n)$ time. The reason is that each recursive call takes $O(1)$ time, and there is one recursive call per node in the tree. But a distinct item is placed at each node, so there are n nodes in total, resulting in an overall $O(n)$ runtime.

3.a. We define Val(m, j, u) to be a function that returns True if one can use j or fewer coins with values any of $v_1, v_2, \ldots, v_m$ to make a total value of u, and returns False otherwise. Value$[m, j, u]$ is an array that stores this Boolean value, for $1 \le m \le n$, $1 \le j \le k$, $1 \le u \le v$.

$$\text{Val}(m,j,u) = \begin{cases} \text{Val}(m, j-1, u-v_m) & m = 1, u \ge v_m \\ \text{Val}(m, j-1, u-v_m) \vee \text{Val}(m-1, j, u) & m > 1, u \ge v_m \\ \text{Val}(m-1, j, u) & m > 1, u < v_m \\ \text{True} & u = 0 \\ \text{False} & m = 1, u < v_m \end{cases}$$

This follows because either one uses an instance of the mth coin or not, with base cases occuring when $m = 1$ or $v_m > u$.

b. There are $(n+1)kv$ possible recursive calls. In an efficient implementation, each of them performs $O(1)$ non-recursive work, yielding a total $O(nkv)$ runtime.

c. For each recursive call we need to record which choice, if any, yields an outcome of True. The choices are: none, use another coin of value v_m, don't use another coin of value v_m.

4. Let $L(i,k) = \sum_{j=i}^{k} |L_j|$, the total length of the ith through k-th lists. Note that $L(i,k) = \text{Lnth}[k] - \text{Lnth}[0]$, where $\text{Lnth}[j] = \sum_{h=1}^{j} |L_h|$. The values Lnth$[0 : n]$ are readily computed in $O(n)$ time.

The recursive formulation is given by:

$$M(i,k) = \begin{cases} \min_{i \le j \le k} \left\{ M(i,j) + M(j+1,k) \right\} + L(i,k) & k > i \\ 0 & k = i \end{cases}$$

The first line follows because the cost of the final merge is $\sum_{j=1}^{k} |L_j|$ regardless of which subsets of lists are being merged in this step, so it suffices to minimize over all choices of j, i.e. of recursively merging $L_i, \ldots, L_j$ and $L_{j+1}, \ldots, L_k$.

b. As $1 \leq i \leq k \leq n$, there are at most $\frac{1}{2}n(n-1)$ recursive calls, or more loosely, $O(n^2)$ recursive calls. In an efficient implementation, the recursive call $M(i,k)$ performs $O(k-i+1) = O(n)$ non-recursive work, Therefore the total runtime is $O(n^3)$.

c. For each recursive call $M(i,k)$, we need to record the value j that yields an optimal outcome for that subproblem.