HW8

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

```
V(n,t)
if n=0 then return 0
else if n>=1 do
  if s[n]>t then V(n-1,t)
  else do
    if not Done[n-1,t] then V(n-1,t);
    if not Done[n-1,t-S[n]] then V(n-1,t-S[n]);
    return max{V(n-1,t),V(n-1,t-S[n])+V[n]}
  end(*else do*)
end(*else if *)
Done[n,t] = true
```

> Assume S is the array storing the size of the items; V is the array storing the values of the items.
>
> No. of the subproblems $(t+1)(n+1) = O(nt)$, because n is from 0 to n, t is from 0 to t.
>
> The cost per subproblem is c, which is $O(1)$, because each function contains a fixed number of checkings.
>
> Hence, the total number of time complexity is $O(nt)$.

2. *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.

```
AugOptBST(i, k)
if i=k+1 then Cost[i,k]←0
else
  Cost[i, k] ← MaxInt;
  RootCost ← Tot[k] − Tot[i − 1];
  for j = i to k do
    if not(Done[i, j − 1]) then AugOptBST(i, j − 1)
    end if
    if not(Done[j + 1, k]) then AugOptBST(j + 1, k)
    end if
    if Cost[i, k]> Cost[i, j − 1] + Cost[j + 1, k] + RootCost then
      Cost[i, k] ← Cost[i, j − 1] + Cost[j + 1, k] + RootCost;
      MinTree[i,k] <- j;
  end for
end if
Done[i, k] ← True
```

```
ProduceBST(i,k,t)
if i=k+1 then t<- null;
else do
  j<-MinTree[i,k];
  t = GetNode(j);
  t.left = produceBST(i, MinTree[i,j-1],t1);
  t.right = produceBST(MinTree[j+1,k],j,t2);
end else
```

> The idea is that we use $MinTree[i,k]$ in the augmented dynamic program to store the root of the tree between $[i,k]$. Therefore, after executing $AugOptBST(1,n)$, $MinTree[1,n]$ will store a root j of the optimal binary search tree. Then its optimal subtrees will have root $MinTree[1,j-1]$ and $MinTree[j+1,n]$. Then construct continuously. In this augmented algorithm, an extra auxiliary Space is $O(n^2)$, but for each subproblem, only $O(1)$ extra time will be added due to the assignment. Hence, the augmented version is still $O(n^3)$.
>
> For the second procedure $ProduceBST(i,k,t)$, t is a return value, which is the tree generated by the $ProduceBST(i,k,t)$. The number of subproblems is n, because each function will be divided into two parts, the left part and the right part. The range $[i:k]$ will cover all of the numbers in $[1:n]$, no overlapping. The cost per subproblem is $O(1)$, only a fixed number of operations are covered. Hence, the total time complexity is $O(n)$

3. *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?

   > a. $SelectK(k,v,n)$ evaluates to True if one can select at most k coins from $v_1, v_2, \ldots, v_n$ to get a total value v and to False otherwise. k is the number of selections left, v is the total value left, and n is the nth type of coin. It is given by the following recursive expression.
   >
   > $$SelectK(k,v,n) = \begin{cases} True & k \geq 0, v = 0 \\ SelectK(k-1, v-v_n, n) \ \vee \ SelectK(k,v,n-1) & k > 0, v > 0, v_n < v \\ SelectK(k,v,n-1) & k \geq 0, v > 0, v_n > v \\ False & Otherwise \end{cases} \quad (1)$$
   >
   > This is correct because the recursion considers choosing from the biggest value whether selecting it or not. This ensures each coin can be selected as the first choice and use $\vee$ to make sure any of the functions meet the condition, the final outcome will be True.
   > b. The k is in range $[0:k]$, v is in range $[0,v]$, and n is in range $[0:n]$. The number of subproblems is $(k+1)(v+1)(n+1) = O(kvn)$. The cost per subproblem is $O(1)$. The total cost is $O(kvn)$.
   >
   > c. For each recursive call we need to record which choice of the coin n, if any, yields an outcome of True.

4. *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?

   > a.

$$MergeList(i,k) = \begin{cases} min_{i \le j \le k} \{MergeList(i, j-1) + MergeList(j,k)\} + \sum_{j=i}^{k} l_j & i \le k \\ 0 & i > k \end{cases} \quad (2)$$

This is correct because the final merge is the merge of two merged lists with a minimal cost plus the cost of the final merge. Then the cost of each subproblem is also generated by this method.

b. The running time of the implementation can be deduced as follows. There are $O(n^2)$ subproblems, as i is in range $[1:n]$, k is in range $[1:n]$. The non-recursive work in the computation of the $MergeList(i,k)$ requires $k-i+1 \le n$ terms, which takes $O(n)$ time. Hence, the total cost will be $O(n^3)$

c. For each recursive call we need to record which choice of the index j, has a minimum cost of the sum of two merged lists.