1. (10 pts) First, write pseudo-code for the branch and bound solution to the knapsack problem (use high-level data structures like priority queues, and represent each partial solution as list of items taken, their total value, and the remaining capacity. Assume the items are numbered from 1 to \( n \) in decreasing order of utility, and that the arrays \( V(1..n) \), \( W(1..n) \), and \( U(1..n) \) contain the values, weights, and utilities of the \( n \) items. Bound the possible value from a partial solution by its current value plus the remaining capacity times the utility of the next item to consider.

Second, simulate your algorithm on the following knapsack instance. Consider the Knapsack problem with capacity 15 and the following items:

<table>
<thead>
<tr>
<th>Item #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>90</td>
<td>48</td>
<td>56</td>
<td>30</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>weight</td>
<td>10</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>util.</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Draw the tree of partial solutions with the current value and remaining capacity for each solution inside the node, and the node’s bound next to the node.

2. (10 pts) Let \( B(n) \) be the number of different binary search trees containing the \( n \) keys \( \{1, 2, \ldots, n\} \). For this problem you will develop a dynamic programming algorithm that, given \( n \), calculates \( B(n) \).

Note that \( B(1) = 1 \) since there is only one one-node binary tree. For two nodes \( B(2) = 2 \) (either node can be root, and there is only one binary search tree consistent with each choice).

(a) (1 pt) Calculate by hand \( B(3) \).

(b) (1 pt) How many \( n = 6 \) key binary trees with key 3 at root (so the left subtree has two nodes, and the right one has 3 nodes) are there?

(c) (3 pts) Construct a recurrence for \( B(n) \), I expect something like

\[
B(n) = \sum_r \text{number of } n\text{-node trees with root } r,
\]

where you need to figure out how many trees there are with root \( r \) in terms of \( B(j) \) values with \( j < n \). Include boundary conditions in your recurrence; what is a convenient boundary value for \( B(0) \)?

(d) (4 pts) Give an iterative (bottom up) algorithm based on the recurrence that computes \( B(n) \) by filling in a table.

(e) (1 pt) What is the running time of your iterative algorithm as a function of \( n \) (use \( \Theta \) notation)?