1. Consider the problem of finding the worst weighted binary search tree for \( n \) items \( e_1 < e_2 < \cdots < e_n \). Suppose item \( e_i \) is accessed \( r_i \) times. The cost of an access to item \( e_i \) is defined to be the path length \( l_i \), measured in nodes, in the binary search tree going from the root to the node storing \( e_i \). Your task is to find a binary search tree \( T \) maximizing the overall access cost, \( \sum_{i=1}^{n} l_i \cdot r_i \). Let \( \text{Cost}(i, k) \) be the cost of the optimal binary search tree storing items \( e_i, e_{i+1}, \ldots, e_k \). Give a recursive formulation for \( \text{Cost}(i, k) \). Then explain how to use it to obtain the desired worst binary search tree. Explain what is the running time of a dynamic programming implementation of this recursive formulation.

2. Consider a variant of the optimal binary search tree for \( n \) items \( e_1 < e_2 < \cdots < e_n \). Each item \( e_i \) is accessed \( r_i \) times. Now, we define the cost of a node \( v \) in a binary search tree \( T \) storing these \( n \) items to be:

\[
\text{Cost}(v) = \sum_{e_i \text{ is an item being stored in the subtree rooted at } v} r_i \cdot \text{depth}(e_i),
\]

where \( \text{depth}(e_i) \) is defined to be the number of nodes on the path from \( v \) to the node storing \( e_i \). The task is to find the tree \( T \) of minimum \( \text{TreeCost} \), where the \( \text{TreeCost} \) is simply the sum of the node costs.

Hint. Figure out what recursive function you need to compute.

3. The assembly line problem.
Suppose there are \( n \) work stations \( W_1, W_2, \ldots, W_n \). You are given a budget of \( \$B \) to spend on possible improvements to some of the workstations. Improving \( W_i \) will cost \( \$b_i \), and will save time \( t_i \). Determine what is the maximum time savings possible with the available budget. You should aim to achieve running time \( O(Bn) \).

4. The vaccine problem.
Suppose there are \( n \) vaccines \( v_1, v_2, \ldots, v_n \). Administering vaccine \( v_i \) to the population will add \( l_i \) life-years to the population, but will cost \( c_i \). You are given a budget of \( \$C \) to spend on these vaccines. Determine what is the maximum possible life-years gain with the available budget. You should aim to achieve running time \( O(Cn) \).

5. Recall the \( \text{Goodness}(v) \) function defined on Page 627, \( \text{GN}(v) \) for short. We modify it, so that the \( i \)th subtree’s Goodness is multiplied by \( i \). To be precise, let \( w_1, w_2, \ldots, w_k \) denote the children of \( v \).

\[
\text{GN}(v) = v.d\text{at} + \sum_{1 \leq i \leq k} i \cdot \text{GN}(w_i).
\]

And for a binary tree, we define

\[
\text{GN}(v) = v.d\text{at} + \text{GN}(v.left) + 2 \cdot \text{GN}(v.right).
\]

You are given an array \( \text{Data}[1:n] \) of integer values, which may include negative values.
a. Suppose the integer values are allocated to the .dat fields of the nodes in an \( n \)-node binary tree \( T \) in postorder. Your task is to determine the tree \( T \) maximizing \( GN(T) \). Give a recursive formulation of the function \( \text{PostBest}(i, k) \), \( \text{PtB}(i, k) \) for short, which equals the largest Goodness value achievable by a \( (k - i + 1) \)-node binary tree for the data values \( \text{Data}[i : k] \). Now explain why a dynamic programming implementation of this function runs in \( O(n^3) \) time.

An example is shown below.

Suppose that \( \text{Data}[1 : 4] = [-2, 4, 1, 3] \). Below we show one possible choice of \( T \):

\[
\begin{array}{c}
\text{.dat values} \\
3 \\
/ \ \\
/ \\
\text{GN values} \\
19 \\
/ \\
/ \\
/ \\
-2 1 \\
/ \\
\text{.dat values} \\
/ \\
/ \\
-2 9 \\
/ \\
/ \\
4 \\
/ \\
/ \\
/ \\
4
\end{array}
\]

b. Repeat part a but now suppose the data values are allocated in preorder. Name your function \( \text{PrB}(i, k) \).

c. Repeat part a but now suppose the data values are allocated in inorder. Name your function \( \text{IB}(i, k) \).

d. Repeat this problem for general trees, with the data values allocated in postorder. Your function \( \text{GPtB} \) will need to have three parameters: \( i, k, l \), which specify that data values \( \text{Data}[1 : k] \) are being stored in a tree whose root has exactly \( l \) subtrees (or it may be more convenient for there to be at most \( l \) subtrees).

e. Repeat this problem for general trees, with the data values allocated in preorder. Name your function \( \text{GPrB}(i, k, l) \).

6. Consider the following 2-player cup game; there is a Blue player and a Red player. There are \( n \) cups \( C_1, C_2, \ldots, C_n \). Each cup holds one or more beads, which can be blue or red, including possibly a mixture of red and blue beads. The players alternate moves, by emptying a consecutive sequence of one or more cups starting from the rightmost non-empty cup. On Blue’s turn, if she empties cups which in combination hold more blue beads, she keeps all the beads in these cups; if not, she has to give the red beads to Red but she still keeps the blue beads. Red plays by the same rules, but switching colors. Your task is to determine how many beads Blue will win if Blue plays first and both Blue and Red play optimally. Suppose that the arrays \( B[1 : n] \) and \( R[1 : n] \) store the number of blue and red beads respectively in each of the cups; i.e. \( B[i] \) is the number of blue beads in \( C_i \) and \( R[i] \) is the number of red beads.

It will be helpful to compute two functions, \( \text{BWin}(k) \), which records the number of beads Blue wins from cups \( C_1, C_2, \ldots, C_k \), if she is first to play, and \( \text{RWin}(k) \), which records the
number of beads Red wins if she is first to play. It will also be useful to compute the values $T[1 : n]$, where $T[k]$ records the total number of beads in the first $k$ cups, namely $C_1, C_2, \ldots, C_k$.

7. Consider the bead game from problem 4, but now Red is a naive player who on each move takes the collection of cups that provide him the most beads on that move. Now determine how many beads Blue wins (a) if Blue plays first, and (b) if Red plays first.

Now suppose that Blue is a kind player who wants to let Red win (if this is possible) but by the smallest amount (i.e. Blue should take as many beads as possible while still losing). Suppose there are $B$ beads in total over all the cups. The goal is to design a dynamic program that runs in time $O(Bn^2)$ to compute how many beads Red ends up taking, supposing that Blue moves first.

Hint. The recursive function you now want to compute is $Btake(k; b)$ which is True if when Red moves first with $k$ cups remaining unemptied, Blue can end up with $b$ beads by some sequence of moves on her part (with Red playing naively, as already defined, on his moves).