Pseudocode for computing optimal BST costs:

computeCost(keys $k_1, ..., k_n$, frequencies f)
    if $(n = 0)$
        return 0
    bestCost = ∞
    for (i = 1...n)
        root = k_i
        cost1 = computeCost($k_1, ..., k_{i-1}$, f)
        cost2 = computeCost($k_{i+1}, ..., k_n$, f)
        cost = cost1 + cost2
        if (cost < bestCost)
            bestCost = cost

    return bestCost + $\sum_{i=1}^{n} f(k_i)$

Suggested implementation order

- Implement the above pseudocode
- Modify it to use memoization (ie check memo before recursive calls... might as well include on/off switch here as well)
- Modify it to actually compute trees (without memoization)
- Modify it to memoize trees (with on/off switch)

I suggest creating trees using pointers, so that you can just point to the root of the optimal children trees (though this could get tricky if ever you need to travel up the tree), but other implementations of trees are fine as well.

There are many options to implement this. Your function could return the optimal BST and cost. Your function could also return nothing and just put its result in the memo, assuming the calling function will check the memo. In both case the switch will turn on or off checking the memo before making the recursive call.

Here is the analysis of the compute cost function. Let $T(n)$ be the number of recursive calls of the compute cost function to find the cost of an optimal BST of size n. Note that this is independent of the actual keys (eg computing the cost takes the same number of recursive calls for any tree of size 3). Also note that when the first key is the root, we make a call to $T(n-1)$, when the second is the root we make calls to $T(1)$ and $T(n-1)$, but when the second to last key is the root we also make calls to $T(n-2)$ and $T(1)$, and when the last

key is the root we make a call to $T(n-1)$. So for every tree of a size we call, we make 2 calls total for every smaller tree size. And so we get

$$\begin{aligned}
T(n) &= 2T(n-1) + 2T(n-2) + ... + 2T(1) \\
&= 2T(n-1) + 2(T(n-2) + T(n-3) + ... + T(1)) \\
&= 2(2T(n-2) + 2T(n-3) + ... + 2T(1)) + 2(T(n-2) + T(n-3) + ... + T(1)) \\
&= 2 \cdot 2(T(n-2) + T(n-3) + ... + T(1)) + 2(T(n-2) + T(n-3) + ... + T(1)) \\
&= 3 \cdot 2(T(n-2) + T(n-3) + ... + T(1)) \\
&= 3 \cdot 2T(n-2) + 3 \cdot 2(T(n-3) + ... + T(1)) \\
&= 3 \cdot 2 \cdot 2(T(n-3) + ... + T(1)) + 3 \cdot 2(T(n-3) + ... + T(1)) \\
&= 3^2 \cdot 2(T(n-3) + ... + T(1)) \\
&... \\
&= 3^{n-2} \cdot 2T(1) \\
&= 3^{n-2} \cdot 2
\end{aligned}$$

$T(1) = 1$ since we only need to make 1 recursive call to find the optimal tree of size 1.

To do the memoization, initialize some sort of data structure (ie the memo) for each key range (eg for keys "A B C D", the memo will have entries of "A", "B", "C","D", "AB", "BC", "CD", "ABC", "BCD"). Each entry is blank initially. Before making a recursive call, check the memo to see if it's there, and only if it's not make the call. Then, right before returning from the call (eg the call for the optimal tree for "AB"), but the result in the memo (eg the cost for "AB" and the optimal tree for "AB"). Two suggestions for memos are a hash and a 2-D array, where the (i, j) cell has the optimal tree and cost for they keys from i to j.