Exercise 6.1  Self-organizing List.

You are given the list K → L → A → N → G → O → P → F → E → R.

a) Execute the following sequence of requests R, E, G, E, N, K, L, O, P, F, E, N on the above list, one time applying the Frequency-count rule and a second time applying the Transpose rule. Count the average number of steps necessary for each of the two cases. Assume that to access the \( i \)-th element of the list, \( i \) steps are necessary. Do not count shifts as extra steps. Which rule is the best one for this example?

b) Now compare the Move-to-front rule, the Frequency-count rule, and the Transpose rule. For each of these rules, specify an access sequence to the above list for which the rule is substantially worse than one of the others. What properties do these sequences have (intuitively)?

Exercise 6.2  Splay Trees & Optimal Search Trees.

Consider the following tree:

a) Give a sequence of insertions (with key values) on an initially empty Splay Tree that leads to the above tree.

b) Give a sequence of 7 keys with corresponding access frequencies, and access frequencies for each interval between two keys, such that the Optimal Search Tree for this sequence has the above structure.

Exercise 6.3  Design of Optimal Search Trees.

Execute the Dynamic Programming algorithm to create an Optimal Search Tree (see Chapter 5.7 of the book) by hand. You are given 5 keys with the following access frequencies:

\[
\begin{array}{c|c|c|c|c|c}
 i/j & 0 & 1 & 2 & 3 & 4 \\
 a_i & 1 & 4 & 7 & 2 & 10 \\
 b_j & 5 & 0 & 0 & 2 & 1 \\
\end{array}
\]
Exercise 6.4  Amortized Analysis.

In this exercise, we consider arrays that can grow dynamically on demand (e.g., `vector` of the C++ standard library). Specifically, we assume to start with an initial array of fixed length $n$, and we insert into it one by one the elements from a given sequence. When the number of values inserted in the array exceeds $n$, a new array of fixed length $k > n$ is defined, and the old array is copied into the new one, where we also place the new element. The creation of an array of length $k$ costs $k$ units of time, and the copying of an element is done in constant time.

a) Describe how to choose $k$ so that each insert operation has amortized constant time, and hence the insertion of $n$ elements can be done in time $\Theta(n)$. Prove that your choice results in constant amortized cost per insert operation using amortized analysis.

b) Now consider the situation where we allow to delete the last element from the array. We allow any mixture of insert and such delete operations. For memory reasons, it may be useful to also shrink the array sometimes. Describe how would you shrink the array, and show that both insertion and removal require amortized constant time.