1. **(Addition of arbitrary precision numbers.)** Consider the following representation of base-10 numbers: the number with (familiar, base-10) digit sequence $d_k d_{k-1} \dots d_1$ is represented as the list $(d_k \dots d_1)$ containing the number's digits (so the most significant digit is given first). Thus, the number 371 is expressed as the list `(3 7 1)`. Define a function `apa-add` which takes two numbers in this representation and returns their sum (in the same representation). (`apa` stands for *arbitrary precision arithmetic*.)

   (Hint: It may simplify your thinking about the problem if you work with the *reverse* of this representation. Note that if n is a (positive, whole) number `(modulo` n `10)` returns the least-significant digit (the "ones" digit) and `(quotient` n `10)` returns the larger digits.)

2. With the same representation as above, define a function `d-multiply`, which multiplies a number in this list representation by a digit (that is, a number in the set $\{0, 1 \dots, 9\}$). Thus, for example, `(d-multiply '(1 2 3) 3)` should return `(3 6 9)`.

3. **(Multiplication of arbitrary precision numbers.)** Note that it is easy to multiply a number by 10 in this representation. Using this fact, and the "multiplication by a digit" definition above, give a recursive procedure to multiply two numbers together.

   (Hint: Multiplication of $d = d_n d_{n-1} \dots d_1$ by $e = e_m e_{m-1} \dots e_1$ can be carried out by the rule

   $$de = d * e_1 + 10 * (d * e_m e_{m-1} \dots e_2).)$$

4. **(Surgery on Binary Search Trees.)** Suppose that T is a binary search tree as shown on the left hand side of Figure 1. As T is a binary search tree, we must have $b < a$; furthermore, all elements in T_1 are smaller than b, all elements in T_2 lie in the range (a, b), and all elements of T_3 are larger than a. Consider now the operation of *right rotation* of T which re-arranges the root and subtrees as shown. It is easy to check that while this "right rotation" changes the structure of the tree, the resulting tree still satisfies the binary search tree property. (The same can be said of "left rotation.")

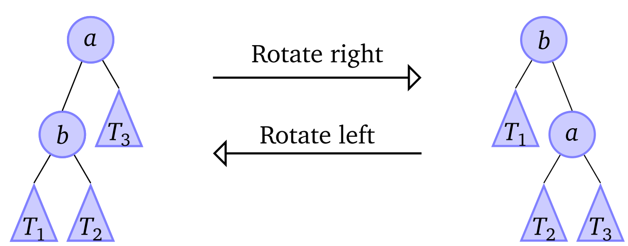

Figure 1: *Right rotation* and *left rotation* of a binary search tree.

The *depth* of an element in a binary tree is the length of the path from the root to the element. It is desirable, for binary search trees, to arrange them so that elements have small depth. (Recall that the *depth* of the tree is the length of the longest path in the tree; equivalently, it is the depth of the "deepest" element.)

Thus, if the depth of T_1 is more than one larger than the depth T_3, a right rotation makes progress toward balancing the tree. Likewise, if the depth of T_3 is more than one more than the depth of T_1, a left rotation seems to be just the thing to improve the tree.

Write a function called tree-repair which takes, as an argument, a binary search tree. tree-repair should use left and right rotations to try to balance the tree T, as follows:

- Recursively repair each of T's two subtrees. (So, you must build a new tree whose two subtrees are obtained by repairing the two subtrees of T.)
- Once the subtrees have been repaired, examine T. If $\text{depth}(T_1) > \text{depth}(T_3) + 1$ return the result of rotating T to the right. If, on the other hand, $\text{depth}(T_3) > \text{depth}(T_1) + 1$ return the result of rotating T to the left.

It would probably be a good idea to start by writing functions that carry out left and right rotations.

5. **(Heapsort.)** Write a sorting procedure which, given a list of numbers, returns a list of the same numbers in sorted order. Your procedure should do the following:

   - Add all of the elements of the initial list into a heap; be careful to arrange your inserts so that the heap remains balanced. (For this purpose, use the heuristic we discussed in class: always insert into the left child and exchange the order of the children.)
   - Repeatedly *extract-min* (extract the minimum element) from the heap, and return the (sorted) list of elements gathered in this way.