Q 1

Define a function `MMERGE` (double 'm' to avoid clash with `LISP`'s same name built-in) that takes a two sorted lists and forms a list consisting of elements in the two lists, again in a sorted order.

Q 2

Implement merge-sort; all you need is to divide the list (use your previous `SPLIT`), merge-sort the parts, and then merge them.

Q 3

Define a function `SHUFFLE` that takes a list and returns a random permutation of the list.

Q 4

Define a fully iterative `PERMUTE`. You may need a usage of `SETF` that we did not do before. With `SETF` you can also set the values of *places*: for instance, `(setf (car lst) 8)` sets the initial element of the list stored in `LST` to 8, here the change is *in place*, the list stored as `LST` changes.

Q 5

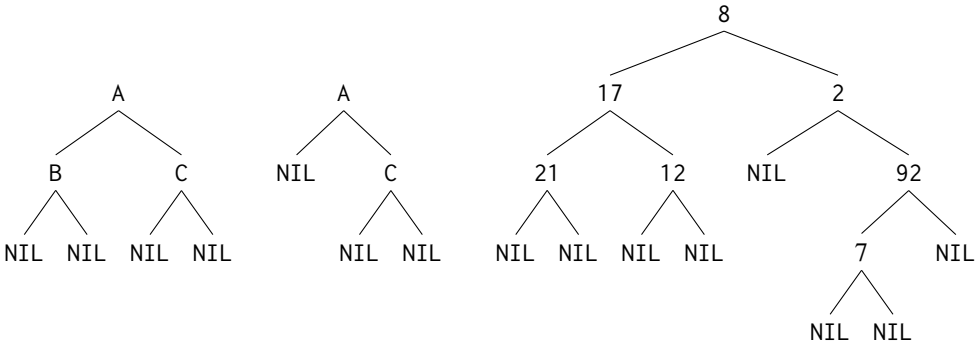
A **binary tree** is defined as follows:

i. `NIL` is a binary tree (an empty one).

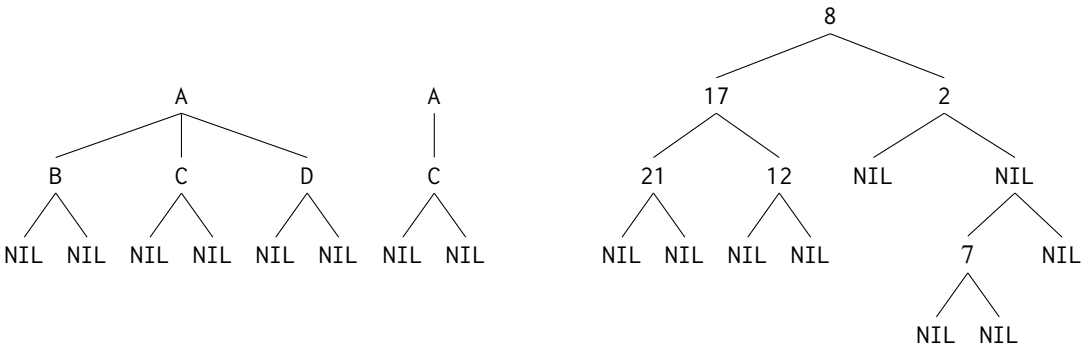
ii. if α is an element (symbol or number) and β and γ are binary trees, then  is a binary tree, where α is the root node, β is the left child (or subtree) and γ is the right child (or subtree).

iii. Nothing else is a binary tree.

here are some example binary trees:



and here are some trees that do not fulfill the definition above:

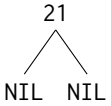


You can represent binary trees as three element LISP lists, where the car is the root element of the tree, cadr is the left child and caddr is the right child. For instance the first two example trees would look like `(A (B NIL NIL) (C NIL NIL))` and `(A NIL (C NIL NIL))`.

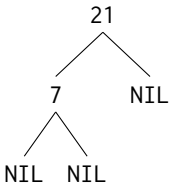
Define a function `binary-tree-p` that gives `T` for a well-formed binary-tree and `NIL` otherwise.

Q 6

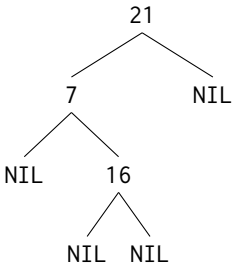
A **binary search tree** is a binary tree such that for a given node α in the tree, all the elements on the left child of α are less than all the elements on the right child of α. You can turn a list of elements into a binary search tree by recursively inserting an element into the tree, observing the binary search tree constraint in each insertion. For instance, you start with the list: `(21 7 16 75 25 12 45 37)` and the empty tree `NIL`. You pick the first element and insert it into this empty tree, obtaining:



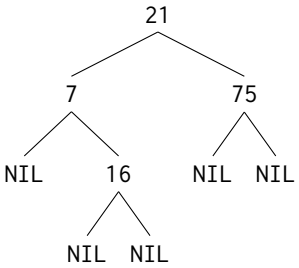
Your list now becomes `(7 16 75 25 12 45 37)`. Then you pick the second element and start travelling in the tree. While traveling you have to stick to the rule that you follow the left path if the element is less than or equal to the elements you encounter, and the right path if not. You do this until you reach an empty tree, in which case you replace the empty tree with a tree whose root is the element in your hand and whose left and right children are empty trees. Now we insert 7 in accord with this and get:



Then, we go on with 16,



Then 75,



and so on.

Define a function that takes a list of numbers and turns it into a binary search tree.