The purpose of this assignment is to continue building some logic programming skills. Please put your code in two files, `bogosort.pl` for question 1 and `trees.pl` for question 2. Submit your files before the beginning of class by copying them to the directory 

/afs/andrew/course/15/317/submit/<userid>/hw08

where `<userid>` is replaced with your Andrew ID.

1. **Bogo sort**

   Prolog can compactly and elegantly represent a sorting algorithm that is even worse than insertion sort. The usual version of Bogo sort works by randomly permuting a list until it becomes sorted. Here, we’ll consider a deterministic variant that searches for a sorted permutation systematically.

   Write a predicate `sorted([integer])` which holds of any \(\leq\)-ordered list of Prolog integers. Then write a predicate `perm([integer], [integer])` which holds whenever its first argument is a permutation of its second. Use these together to implement a concise version of deterministic bogo sort as a predicate `bogosort([integer], [integer])` that searches for a sorted permutation of its first argument. Your predicate should be usable as a sorting function, at least on very small inputs.

2. **Binary Trees**

   Binary trees can be represented in Prolog by two function symbols\(^1\): `node(Val, Left, Right)` where `Val` is the value associated with the node, `Left` is the left subtree and `Right` is the right subtree, and `leaf` representing the empty tree.

   (a) Define a type predicate `binary_tree/1` that is valid when the argument is a binary tree.

   A binary tree can be traversed in three ways: preorder, inorder, and postorder. In preorder traversal the root is visited first, then the left subtree is traversed, and finally the right subtree is traversed. In inorder traversal the left subtree is traversed first, then the root is visited, and then the right subtree is traversed. In postorder traversal, the order is left subtree, right subtree, and finally the root is visited.

---

\(^1\) Also called functors in the terminology of Prolog
(b) Define predicates `preorder/2`, `inorder/2`, `postorder/2` that hold when the first argument is a binary tree and the second argument is the list obtained by traversing the tree in the corresponding order.

A *binary search tree* (BST) is a binary tree with integer values in the nodes with the property that for any node in the tree, the left subtree contains only values that are smaller than the value of the node and the right subtree contains only values that are greater than the value of the node. Note that inorder traversal of the nodes of a BST visits the nodes in increasing order.

(c) Define a type predicate `bst/1` that is valid when the argument is a BST with integer values in the nodes.

(d) Define a predicate `bst_insert/3` such that `bst(Val, Tree1, Tree2)` is valid when `Tree2` is obtained by inserting `Val` in `Tree1`. If `Tree1` is a BST, then `Tree2` should also be a BST.

(e) Define a predicate `bst_sort/2` that sorts a list by first constructing a BST with the values of the list and then traversing it inorder.