Trees
1. Write (square-tree tree), which returns the same tree structure, but with every element squared. Don’t use map!

   (define (square-tree tree)
     (make-tree (square (datum tree))
       (square-forest (children tree))))

   (define (square-forest forest)
     (if (null? forest)
       '()
       (cons (square-tree (car forest)) (square-forest (cdr forest)))))

2. Write (max-of-tree tree) that does the obvious thing. The tree has at least one element.

   (define (max-of-tree tree)
     (if (null? (children tree))
       (datum tree)
       (max (datum tree) (max-of-forest (children tree)))))

   (define (max-of-forest forest)
     (if (null? (cdr forest))
       (max-of-tree (car forest))
       (max (max-of-tree (car forest)) (max-of-forest (cdr forest))))))

3. Write (listify-tree tree) that turns the tree into a list in any order. (This one you can’t use map even if you tried... Muwahahaha)

   (define (listify-tree tree)
     (cons (datum tree) (listify-forest (children tree))))

   (define (listify-forest forest)
     (if (null? forest)
       '()
       (append
         (listify-tree (car forest)) (listify-forest (cdr forest))))))

4. A maximum heap is a tree whose children’s data are all less-than-or-equal-to the roots datum. Of course, its children are all maximum heaps as well. Write (valid-max-heap? tree) that checks if this is true for a given tree.

   (define (valid-max-heap? tree)
     (and (= (datum tree) (max-of-tree tree))
       (valid-max-heaps? (children tree))))

   (define (valid-max-heaps? forest)
     (cond ((null? forest) #t)
       (else (and (valid-max-heap? (car forest))
                   (valid-max-heaps? (cdr forest))))))

Binary Search Trees
1. Jimmy the Smartass was told to write (valid-bst? bst) that checks whether a tree satisfies the binary-search-tree property elements in left subtree are smaller than datum, and elements in right subtree are larger than datum. He came up with this:

   (define (valid-bst? bst)
     (cond ((null? bst) #t)
       (else (and (or (null? (left-branch bst))
                         (and (< (datum (left-branch bst)) (datum bst))
                           (valid-bst? (left-branch bst))))
                         (or (null? (right-branch bst))
                           (and (> (datum (right-branch bst)) (datum bst))
                             (valid-bst? (right-branch bst))))))))
Why will Jimmy never succeed in life? Give an example that would fool his pitiful procedure.

Checking if the bst property is true for your immediate children does not guarantee that everything in one subtree will satisfy the bst property. For instance, if you have a root 10, a left child 3, and the right child of 3 is 12, Jimmy’s procedure will say true when it is false.

2. Write (sum-of bst) that takes in a binary search tree, and returns the sum of all the data in the tree.

```
(define (sum-of bst)
  (cond ((null? bst) 0)
        (else (+ (datum bst) (sum-of (left-branch bst))
                 (sum-of (right-branch bst))))))
```

3. Write (max-of bst) that takes in a binary search tree, and returns the maximum datum in the tree. The tree has at least one element.

```
(define (max-of bst)
  (cond ((null? (right-branch bst)) (datum bst))
        (else (max-of (right-branch bst))))
```

4. Write (listify bst) that converts elements of the given bst into a list. The list should be in NON-DECREASING ORDER!

```
(define (listify bst)
  (cond ((null? bst) ’())
        (else (append (listify (left-branch bst))
                     (list (datum bst)
                          (listify (right-branch bst))))))
```

5. Write (remove-leaves bst) that takes in a bst and returns the bst with all the leaves removed.

```
(define (remove-leaves bst)
  (cond ((null? bst) ’())
        ((leaf? bst) ’())
        (else (make-tree (datum bst)
                         (remove-leaves (left-branch bst))
                         (remove-leaves (right-branch bst))))))
```

6. Write (height-of tree) that takes in a tree and returns the height the length of the longest path from the root to a leaf.

```
(define (height-of tree)
  (cond ((leaf? tree) 0)
        (else (+ 1 (max (height-of (left-branch tree))
                         (height-of (right-branch tree))))))
```