Exercise 2.63.

Each of the following two procedures converts a binary tree to a list.

```
(define (tree->list-1 tree)
  (if (null? tree)
      '()
      (append (tree->list-1 (left-branch tree))
              (cons (entry tree)
                    (tree->list-1 (right-branch tree))))))

(define (tree->list-2 tree)
  (define (copy-to-list tree result-list)
    (if (null? tree)
        result-list
        (copy-to-list (left-branch tree)
                      (cons (entry tree)
                            (copy-to-list (right-branch tree)
                                          result-list)))))
  (copy-to-list tree '()))
```

a. Do the two procedures produce the same result for every tree? If not, how do the results differ? What lists do the two procedures produce for the trees in figure 2.16?

b. Do the two procedures have the same order of growth in the number of steps required to convert a balanced tree with n elements to a list? If not, which one grows more slowly?

Answer.

a. To see is to believe! We can determine their performance by running tests on the trees in figure 2.16:

```
(define Tree-A (make-tree 7
                          (make-tree 3
                                     (make-tree 1 '() '())
                                     (make-tree 5 '() '()))
                          (make-tree 9
                                     '()
                                     (make-tree 11 '() '()))))

(define Tree-B (make-tree 3
                          (make-tree 1 '() '())
                          (make-tree 7
                                     (make-tree 5 '() '())
                                     (make-tree 9
                                                '()
                                                (make-tree 11 '() '())))))

(define Tree-C (make-tree 5
                          (make-tree 3
                                     (make-tree 1 '() '())
                                     '())
                          (make-tree 9
                                     (make-tree 7 '() '())
                                     (make-tree 11 '() '()))))

(tree->list-1 Tree-A)
;Value 20: (1 3 5 7 9 11)

(tree->list-2 Tree-A)
;Value 21: (1 3 5 7 9 11)
```

```
(tree->list-1 Tree-B)
;Value 22: (1 3 5 7 9 11)

(tree->list-2 Tree-B)
;Value 23: (1 3 5 7 9 11)

(tree->list-1 Tree-C)
;Value 24: (1 3 5 7 9 11)

(tree->list-2 Tree-C)
;Value 25: (1 3 5 7 9 11)
```

Now we see that both procedures produce the same list for every tree.

b. Suppose we want to flatten a balanced binary tree of n nodes into a list using these two procedures seperately. We also assume that the total time costed by these two procedures are $T_1(n)$ and $T_2(n)$ respectfully. Since the `tree->list-1` procedure appends the resulting flattened left branch to the flattened right branch together with the entry of the tree. Thus, both the left branch and the right branch flatten a tree of approximate $n/2$ nodes. On the other hand, we know that the number of steps to perform `append` operation has an order of growth $\Theta(n)$, where n is the number of elements in the first argument list, here it is the flattened left branch which contains nearly $n/2$ elements. Hence, for `tree->list-1` we have:

$$T_1(n) = 2 \times T_1\left(\frac{n}{2}\right) + \Theta\left(\frac{n}{2}\right)$$

Solve the above equation reveals

$$T_1(n) = \Theta(n \log n)$$

The `tree->list-2` procedure flattens a tree into a list by `cons`ing the left most element onto the flattened left branch as well as `cons`ing the entry element onto the flattened right branch. Since the number of steps takes up by `cons` only has an order of growth $\Theta(1)$, we have:

$$T_2(n) = 2 \times T_2\left(\frac{n}{2}\right) + \Theta(1)$$

Solve this equation we get

$$T_2(n) = \Theta(n)$$

Therefore, the steps required by `tree->list-2` grows more slowly as the size of the problem increase.