

For the next few problems, when testing procedures on trees, you may choose to “cheat” and type in the tree at the top of page 251 directly (circumventing the ADTs `make-nonempty-tree` and `make-empty-tree`):

```
(define tree '(4 (2 (1 () ())) (3 () ())) (6 (5 () ())) (7 () ())))
```

- 8a:** [HKK 8.4 expanded] Eliminate the call to `append` by using an “onto” parameter to make a more efficient version of `inorder`. Explain why the new version is more efficient.
- 8b:** [HKK 8.30] Fill in the following definition of the procedure `successor-of-in-or`. This procedure should take three arguments: a value (*value*), a binary search tree (*bst*), and a value to return if no element of the tree is larger than *value* (*if-none*). If there is any element, *x*, of *bst* such that  $x > \textit{value}$ , the smallest such element should be returned. Otherwise, *if-none* should be returned.

```
(define successor-of-in-or
  (lambda (value bst if-none)
    (cond ((empty-tree? bst)
           _____)
          ((<= (root bst) value)
           (successor-of-in-or _____
                               _____))
          (else
           (successor-of-in-or _____
                               _____))))
```

- 8c:** [HKK 8.31] Write a procedure that takes as arguments a binary search tree of numbers, a lower bound, and an upper bound and counts how many elements of the tree are greater than or equal to the lower bound and less than or equal to the upper bound. Assume that the tree may contain duplicate elements. Make sure your procedure doesn’t examine more of the tree than is necessary.