演習問題 5.7

> ソート関数がソート済みリストに対し $O(n)$ 時間しかかからないことを示す。

```
sort :: Ord a => [a] -> [a]
sort = inOrder [] . construct

construct :: Ord a => [a] -> SplayHeap a
construct = foldl (flip insert) empty

inOrder :: [a] -> SplayHeap a -> [a]
inOrder xs E = xs
inOrder xs (T a x b) = inOrder (x:bs) a where
  bs = inOrder xs b
```

`construct` がソート済みリストに対し $O(n)$ 時間で動作することを示すため、ソート済みリストから作られたスプレーヒープへの `insert` が $O(1)$ 時間で動作することを示そう。

ここでは、ソートは全て昇順ソートであるとする。

空ヒープへ挿入する際、要素 x を挿入すると `T E x E` となる。

昇順にヒープへの挿入を行う場合、木の最右ノードが `E` であるため、`partition` では以下のパターンにマッチする。

```
partition pivot t@(T a x b) =
  if x <= pivot         -- 昇順なので常にtrue
    then case b of
      E -> (t, E)       -- 常にこちらにマッチ
      T b1 y b2 -> ...
```

このパターンにマッチするとき、木の最右ノードは常に `E` となる。

そのため、続けて昇順にヒープへの挿入を行う場合、常にこのパターンにマッチすることとなる。

このパターンにおける処理は $O(1)$ 時間で動作するため、 `insert` は $O(1)$ 時間で動作する。

また、 `inOrder` は呼び出し毎に1つずつノードを消費するため、 $O(n)$ 時間で動作する。

以上から、 `sort` はソート済みリストに対し $O(n)$ 時間で動作する。

メモ: ソート済みリストに対する `construct` の挙動

```
>>> mapM_ (print . construct) [[1..x] | x <- [0..5]]
E
T E 1 E
T (T E 1 E) 2 E
T (T (T E 1 E) 2 E) 3 E
T (T (T (T E 1 E) 2 E) 3 E) 4 E
T (T (T (T (T E 1 E) 2 E) 3 E) 4 E) 5 E
```