演習問題 4.2

`take k (sort xs)` が $O(n \cdot k)$ で完了することを示す。（例を眺めながら雰囲気を示す）

```
> take 2 (sort (Cons 3 (Cons 1 (Cons 2 Nil))))
= take 2 (isort Nil (Cons 3 (Cons 1 (Cons 2 Nil))))        -- 1. sort xs
= take 2 (isort (insert 3 Nil) (Cons 1 (Cons 2 Nil)))      --    isort ys (Cons x xs)
= take 2 (isort (insert 1 (insert 3 Nil)) (Cons 2 Nil))    --    isort ys (Cons x xs)
= take 2 (isort (insert 2 (insert 1 (insert 3 Nil))) Nil)  --    isort ys (Cons x xs)
= take 2 (insert 2 (insert 1 (insert 3 Nil)))              --    isort ys Nil
= take 2 (insert 2 (insert 1 (Cons 3 Nil)))                -- 2. insert x Nil
= take 2 (insert 2 (Cons 1 (Cons 3 Nil)))                  --    insert x (Cons y ys) (then)
= take 2 (Cons 1 (insert 2 (Cons 3 Nil)))                  --    insert x (Cons y ys) (else)
= Cons 1 (take 1 (insert 2 (Cons 3 Nil)))                  -- 3. take n (Cons x xs)
= Cons 1 (take 1 (Cons 2 (Cons 3 Nil)))                    -- 2. insert x (Cons y ys) (then)
= Cons 1 (Cons 2 (take 0 (Cons 3 Nil)))                    -- 3. take n (Cons x xs)
= Cons 1 (Cons 2 Nil)                                      --    take 0 _
```

`take` `sort` `isort` `insert` は全て遅延評価される。これらの関数呼び出しが構成するサンクでは、可能な限りサンクの外側からパターンマッチが実施される。

1. `take k (sort xs)` の評価では、まず `sort` が展開され、さらに `isort` が再帰的に展開されることで、`insert` によるサンクを構成する。
2. 次に、最も内側の `insert` がパターンマッチし、再帰的な `insert` の展開によって最小の数が `take` がマッチ可能な位置に来る。（これはバブルソートのような値の伝搬に見える。）
3. それから、`take` がパターンマッチし、ストリームの先頭が確定する。

あとは、`take 0 _` または `take _ Nil` となるまで 2. と 3. を繰り返す。

このとき、take の k が一回消費されるまでに実行される insert は $O(n)$ 回なので、`take k (sort xs)` は $O(n \cdot k)$ で完了する。

付録: `sort` の定義

```
sort :: Ord a => Stream a -> Stream a
sort = isort Nil where
  isort ys Nil = ys
  isort ys (Cons x xs) = isort (insert x ys) xs
  insert x Nil = Cons x Nil
  insert x (Cons y ys) = if x <= y
    then Cons x (Cons y ys)
    else Cons y (insert x ys)

data Stream a = Nil | Cons a (Stream a)
```