Making a quadtree balanced in linear time. The algorithm processes the tree nodes level by level. We will incrementally assign to each node u four *neighbor pointers* u.N, u.W, u.S, u.E, pointing to four tree nodes v whose squares are adjacent to u at the respective side. The side length $s(v)$ of the target of each pointer satisfies the relation

$$s(v) = \begin{cases} s(u), \\ s(u) \times 2, \text{ or} \\ s(u) \times 4 \text{ (a } \textit{bad} \text{ pointer).} \end{cases}$$

A pointer that does not satisfy one of these equations is *illegal*. Pointers at the boundary of the bounding box are null (and are regarded as good pointers). As soon as every node in the whole tree has four good neighbor pointers, the tree must be balanced.

At the beginning of stage t, precisely all tree nodes u up to depth t have neighbor pointers, and there are no bad pointers. The global procedure is the following loop:

for $t = 0, 1, 2, \ldots$
  for every node u at level t:
    if u is not a leaf:
      set-pointers(u)

The procedure *set-pointers*(u) sets the neighbor pointers of u's children as follows:

procedure *set-pointers*(u):
  { At the beginning, if u is at level ℓ of the tree, all tree nodes at level up to and including ℓ have only good pointers. }
  for $v = u$.NE, u.NW, u.SW, u.SE:
    Set each pointer v.N, v.W, v.S, v.E to the corresponding pointer from u or to the appropriate sibling node of v.
  Let L be the list of bad pointers that have been created.
  for each pointer in L:
    { We will subdivide the target square of the pointer if necessary, and replace the bad pointer by a good pointer. }
    More precisely, assume for example that the bad pointer is v.N $= w$.
    { v is at level $\ell + 1$, and w is at level $\ell - 1$ }
    if w is a leaf:
      Refine w into four subsquares w.NE, w.NW, w.SW, w.SE.
      set-pointers(w)
    Set v.N := w.SW or w.SE, as appropriate.
  { At the end, all nodes at level up to ℓ have only good pointers. }

Exercises. 1. (16 pointers) Spell out the details how the sixteen neighbor pointers of the four children of u are set.

2. (4 pointers) Show that L can contain at most 4 bad pointers.

Correctness. By the assumption that holds at the beginning, the neighbor pointers from u are good, and hence the newly created neighbor pointes from the children v of u can be bad but not illegal. All bad pointers that are created are replaced by good pointers before the procedure *set-pointers* returns.

The algorithm never refines a square if it is not necessary. The runtime is proportional to the number of tree nodes that exist plus the number of tree nodes that are created.

Variations. The recursion can be replaced by a single stack that contains all bad pointers. Some further streamlining is possible.