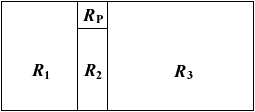
The key insight that leads to algorithms for ordered treemaps is that it is possible to create a layout in which items that are next to each other in the given order are adjacent in the treemap. Although such a layout does not follow the simple linear order of the slice-and-dice layout, it provides useful cues for locating objects and turns out to provide constraints on the layout that discourage large discontinuous changes with dynamic data.

We discuss two closely related algorithms for creating layouts that approximately preserve order. Both follow a similar recursive process, starting with a rectangle *R* to be subdivided.� The input is a list of items that are ordered by an index and have varied areas.� A particular item (the pivot) is chosen and placed at the side *R* as a square (aspect ratio = 1). The remaining items in the list are placed in three large rectangles that make up the rest of the display area. The algorithm is then applied recursively to each of these rectangles.

In the first algorithm, pivot-by-size, the pivot is taken to be the item with the largest area. The motivation for this choice is that the largest item will be the most difficult to place, so it should be done first. The algorithm, as illustrated in Fig. 3, can be described as follows:



**Figure 3. The pivot configuration.**

1.        Let *P*, the pivot, be the item with the largest area in the list of items.

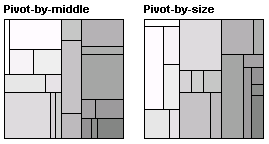
2.        If the width of *R* is greater than or equal to the height, divide R into four rectangles, *R*1, *R*P, *R*2, and *R*3 as shown in Fig. 2. (If the height is greater than the width, use the same basic arrangement but flipped along the line *y=x*.)

3.        Place *P* in the rectangle *RP*, whose exact dimensions will be determined in Step 4.

4.        Divide the items in the list, other than *P*, into three lists, *L*1, *L*2, and *L*3, to be laid out in *R*1, *R*2, and *R*3. *L*1 and *L*3 all may be empty lists. (Note that the contents of these three lists completely determine the placement of the rectangles in Figure 3.) Let *L*1 consist of all items whose index is less than P in the ordering. Let *L*2 and *L*3 be such that all items in *L*2 have an index less than those in *L*3, and the aspect ratio of *P* is as close to 1 as possible. We add the proviso, to avoid degenerate layouts, that *L*3 cannot contain exactly one item.

5.        Recursively lay out *L*1, *L*2, and *L*3 (if any are non-empty) in *R*1, *R*2, and *R*3 according to this algorithm.

The second ordered treemap algorithm, pivot-by-middle, is almost identical except that the pivot is taken to be the middle item of the list - that is, if the list has n items, the pivot is item number n/2, rounded down. The motivation behind this choice is that it is likely to create a balanced layout. In addition, because the choice of pivot does not depend on the size of the items, the layouts created by this algorithm may not be as sensitive to changes in the data as pivot by size. Figure 4 shows examples of the layouts created by the two algorithms.



**Figure 4. Pivot layouts. Shading indicates order, which is roughly preserved.**

Both algorithms have the property that they create layouts that roughly preserve the ordering of the index of the items, which will fall in a left-to-right and top-to-bottom direction in the layout. The two algorithms are also reasonably efficient: pivot-by-size has performance characteristics similar to QuickSort (order *n*log *n* average case and *n*2 worst case) while pivot-by-middle has order� *n*log *n* performance in the worst case.

Although the two algorithms produce layouts with relatively low aspect ratios (as described in the following sections) they are not optimal in this regard. The stipulations in step 4 of the algorithm avoid some but not all degenerate layouts with high aspect ratios, so we experimented with post-processing strategies designed to improve the layout aspect ratio. For example we tried adding a last step to the algorithm in which any rectangle that is divided by a segment parallel to its longest side is changed so that it is divided by a segment parallel to its shortest side. Because this step gave only a small improvement in layout aspect ratio while dramatically decreasing layout stability, we did not include it in the final algorithm.