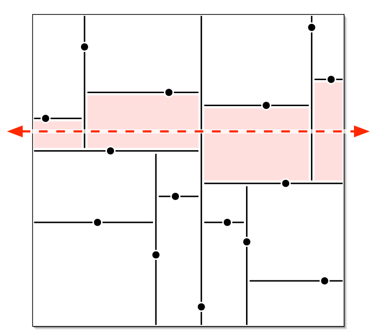

- For this and all future homeworks, groups of up to three students can submit (or present) a single common solution. Please remember to write the names of all group members on every page.
- **Please fill out the online input survey linked from the course web page no later than Thursday, January 28.** Among other things, this survey asks you to identify the other members of your HW1 group, so that we can partition the class into presentation clusters without breaking up your group. We will announce the presentation clusters on Friday, January 29.
- Students in **Cluster 1** will present their solutions to Jeff or one of the TAs, on Tuesday or Wednesday of the due date (February 2 or February 3), instead of submitting written solutions. **Each homework group in Cluster 1 must sign up for a 30-minute time slot no later than Monday, February 1.** Signup sheets will be posted at 3303 Siebel Center ('The Theory Lab') later this week. Please see the course web page for more details.

---

1. Suppose we have n points scattered inside a two-dimensional box. A *kd-tree* recursively subdivides the points as follows. First we split the box into two smaller boxes with a *vertical* line, then we split each of those boxes with *horizontal* lines, and so on, always alternating between horizontal and vertical splits. Each time we split a box, the splitting line partitions the rest of the interior points *as evenly as possible* by passing through a median point in the interior of the box (*not* on its boundary). If a box doesn't contain any points, we don't split it any more; these final empty boxes are called *cells*.



A kd-tree for 15 points. The dashed line crosses the four shaded cells.

   (a) How many cells are there, as a function of n? Prove your answer is correct.

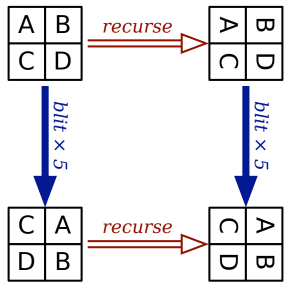
   (b) In the worst case, *exactly* how many cells can a horizontal line cross, as a function of n? Prove your answer is correct. Assume that $n = 2^k - 1$ for some integer k.

   (c) Suppose we have n points stored in a kd-tree. Describe and analyze an algorithm that counts the number of points above a horizontal line (such as the dashed line in the figure) as quickly as possible. *[Hint: Use part (b).]*

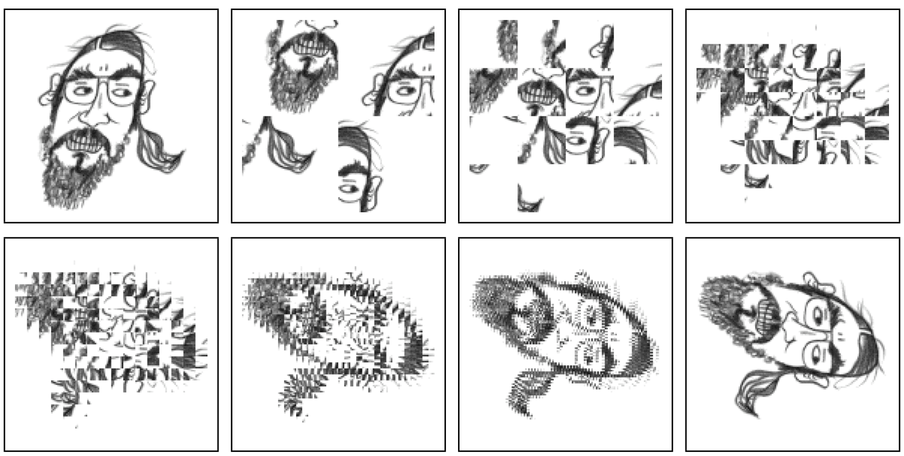
   (d) Describe an analyze an efficient algorithm that counts, given a kd-tree storing n points, the number of points that lie inside a rectangle R with horizontal and vertical sides. *[Hint: Use part (c).]*

2. Most graphics hardware includes support for a low-level operation called *blit*, or **bl**ock **t**ransfer, which quickly copies a rectangular chunk of a pixel map (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

   Suppose we want to rotate an $n \times n$ pixel map 90° clockwise. One way to do this, at least when n is a power of two, is to split the pixel map into four $n/2 \times n/2$ blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we could *first* recursively rotate the blocks and *then* blit them into place.



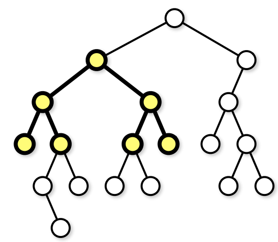
Two algorithms for rotating a pixel map.
Solid arrows indicate blitting the blocks into place; hollow arrows indicate recursively rotating the blocks.



The first rotation algorithm (blit then recurse) in action.

   (a) Prove that both versions of the algorithm are correct when n is a power of two.
   (b) *Exactly* how many blits does the algorithm perform when n is a power of two?
   (c) Describe ***briefly*** how to modify the algorithm so that it works for arbitrary n, not just powers of two. Pseudocode is not required. How many blits does your modified algorithm perform?
   (d) What is your algorithm's running time if a $k \times k$ blit takes $O(k^2)$ time?
   (e) What if a $k \times k$ blit takes only $O(k)$ time?

3. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.