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.](image)

(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 \textit{blit}, or block transfer, 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 \textit{memcpy()}.

Suppose we want to rotate an $n \times n$ pixel map $90^\circ$ 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.

\begin{center}
\begin{tabular}{c}
\begin{tabular}{c}
A B \\
C D \end{tabular}
\end{tabular}
\end{center}
\begin{center}
\begin{tabular}{c}
\begin{tabular}{c}
C A \\
D B \end{tabular}
\end{tabular}
\end{center}
\begin{center}
\begin{tabular}{c}
\begin{tabular}{c}
\text{recurse}
\end{tabular}
\end{tabular}
\end{center}

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

(a) Prove that both versions of the algorithm are correct when $n$ is a power of two.
(b) \textit{Exactly} how many blits does the algorithm perform when $n$ is a power of two?
(c) Describe how to modify the algorithm so that it works for arbitrary $n$, not just powers of two. 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.

![Binary Tree Diagram]

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