

It's not hard to analyze the running time of this program: each invocation of the inner loop, over i , takes $O(n)$ time, and this inner loop is invoked $O(n)$ times total, so the overall running time is $O(n^2)$.

The trouble is, for the large values of n they're working with, the program takes several minutes to run. On the other hand, their experimental setup is optimized so that they can throw down n particles, perform the measurements, and be ready to handle n more particles within a few seconds. So they'd really like it if there were a way to compute all the forces F_j much more quickly, so as to keep up with the rate of the experiment.

Help them out by designing an algorithm that computes all the forces F_j in $O(n \log n)$ time.

5. *Hidden surface removal* is a problem in computer graphics that scarcely needs an introduction: when Woody is standing in front of Buzz, you should be able to see Woody but not Buzz; when Buzz is standing in front of Woody, . . . well, you get the idea.

The magic of hidden surface removal is that you can often compute things faster than your intuition suggests. Here's a clean geometric example to illustrate a basic speed-up that can be achieved. You are given n nonvertical lines in the plane, labeled L_1, \dots, L_n , with the i^{th} line specified by the equation $y = a_i x + b_i$. We will make the assumption that no three of the lines all meet at a single point. We say line L_i is *uppermost* at a given x -coordinate x_0 if its y -coordinate at x_0 is greater than the y -coordinates of all the other lines at x_0 : $a_i x_0 + b_i > a_j x_0 + b_j$ for all $j \neq i$. We say line L_i is *visible* if there is some x -coordinate at which it is uppermost—intuitively, some portion of it can be seen if you look down from “ $y = \infty$.”

Give an algorithm that takes n lines as input and in $O(n \log n)$ time returns all of the ones that are visible. Figure 5.10 gives an example.

6. Consider an n -node complete binary tree T , where $n = 2^d - 1$ for some d . Each node v of T is labeled with a real number x_v . You may assume that the real numbers labeling the nodes are all distinct. A node v of T is a *local minimum* if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge.

You are given such a complete binary tree T , but the labeling is only specified in the following *implicit* way: for each node v , you can determine the value x_v by *probing* the node v . Show how to find a local minimum of T using only $O(\log n)$ *probes* to the nodes of T .

7. Suppose now that you're given an $n \times n$ grid graph G . (An $n \times n$ grid graph is just the adjacency graph of an $n \times n$ chessboard. To be completely precise, it is a graph whose node set is the set of all ordered pairs of

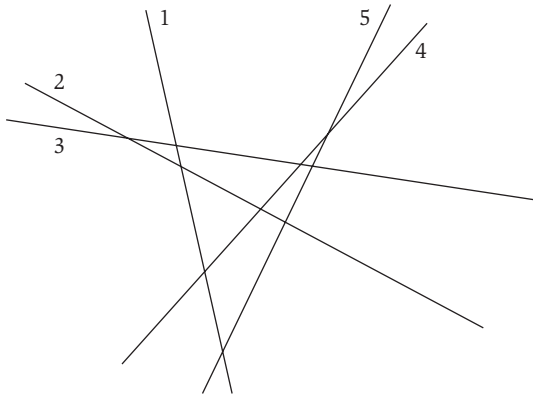


Figure 5.10 An instance of hidden surface removal with five lines (labeled 1–5 in the figure). All the lines except for 2 are visible.

natural numbers (i, j) , where $1 \leq i \leq n$ and $1 \leq j \leq n$; the nodes (i, j) and (k, ℓ) are joined by an edge if and only if $|i - k| + |j - \ell| = 1$.)

We use some of the terminology of the previous question. Again, each node v is labeled by a real number x_v ; you may assume that all these labels are distinct. Show how to find a local minimum of G using only $O(n)$ probes to the nodes of G . (Note that G has n^2 nodes.)

Notes and Further Reading

The militaristic coinage “divide and conquer” was introduced somewhat after the technique itself. Knuth (1998) credits John von Neumann with one early explicit application of the approach, the development of the Mergesort Algorithm in 1945. Knuth (1997b) also provides further discussion of techniques for solving recurrences.

The algorithm for computing the closest pair of points in the plane is due to Michael Shamos, and is one of the earliest nontrivial algorithms in the field of computational geometry; the survey paper by Smid (1999) discusses a wide range of results on closest-point problems. A faster randomized algorithm for this problem will be discussed in Chapter 13. (Regarding the nonobviousness of the divide-and-conquer algorithm presented here, Smid also makes the interesting historical observation that researchers originally suspected quadratic time might be the best one could do for finding the closest pair of points in the plane.) More generally, the divide-and-conquer approach has proved very useful in computational geometry, and the books by Preparata and Shamos