⟨Problem 84-2⟩ by J. L. Bentley (Bell Laboratories), C. E. Leiserson (MIT), R. L. Rivest (MIT), and C. J. van Wyk

Given $2n$ distinct endpoints of n chords on the unit circle, count the number of intersections between chords (if k chords intersect at one point, that point counts as $\binom{k}{2}$ intersections). What is the complexity of this problem as a function of n?

Solution by the proposers

Label the endpoints with the integers 1 to n such that two points have the same label if and only if they are endpoints of the same chord. Let σ be the string obtained by reading the labels in the order in which they appear around the circle. Note that σ will contain $2n$ labels, and each label will appear exactly twice. The chord labeled i and the chord labeled j intersect if and only if i appears exactly once in σ between the two instances of j.

The idea behind the algorithm is to scan σ once. When a label i is seen for the first time, an interval is opened for it. When i is seen again, we close its interval and find out how many intervals have been opened and not closed since i was first seen. Each of these represents an intersection between chord i and some other chord. For example, if $\sigma = 1\ 2\ 3\ 3\ 4\ 2\ 4\ 1$, we would open intervals for 1, 2, and 3, then close the interval for 3 (noting no intersections), open an interval for 4, close the interval for 2 (counting one intersection), and finally close the intervals for 4 and 1 (noting no more intersections). Notice that when we close the interval for 2 we do not count all open intervals (1 and 4), but only the one that opened after 2. In Fig. 2, each arc represents an interval.

In the algorithm the intervals are represented by two arrays of length n. Array E stores the locations of the first endpoints of chords, while *seen* records that an endpoint of a chord has already been seen. The algorithm also represents a set of S of endpoint locations using a range query data structure that permits us to insert and delete items, and to answer the question "How many elements in S are larger than x?" in time $O(\log n)$ when there are n items in S (Exercise 6.2.3-15 in [1]).

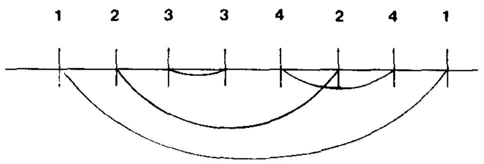

FIGURE 2

```
$i \leftarrow 1$
$intersections \leftarrow 0$
$S \leftarrow \emptyset$
while ($i \leqslant 2n$) {
        $j \leftarrow \sigma_i$
        if ($seen_j = FALSE$) {/* open an interval for $j$ */
                $seen_j \leftarrow TRUE$
                $E_j \leftarrow i$
                $S \leftarrow S \cup \{i\}$
        } else { /* close the interval for $j$ and count intersections */
                $intersections \leftarrow intersections + |\{x | x \in S \wedge x > E_j\}$
        }
        $i \leftarrow i + 1$
        $S \leftarrow S - \{E_j\}$
}
```

Upon completion, the value of the variable *intersections* is the number of intersections. For each label i let $\sigma_{s_i} = \sigma_{t_i} = i$ with $s_i < t_i$ (s_i and t_i index the left and right endpoints of chord i). Consider two chords i and j with $s_i < s_j$. If chords i and j cross, then $s_j < t_i < t_j$; the intersection will be counted exactly once, namely, when σ_{t_i} is examined. If chords i and j do not cross, then $t_j < t_i$, so no intersection will be counted between chords i and j.

To analyze the algorithm's running time, notice that each label is inserted once into S, and one range query may be made about it before it is deleted from S. Since each of these operations takes time $O(\log n)$, the running time of the algorithm is $O(n \log n)$. (The algorithm can easily be modified to report all intersections between chords in time $O(n \log n + m)$ when there are m intersections: when an interval i is closed, report all open intervals that opened after i was opened.)

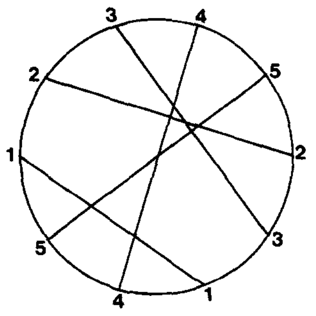

FIGURE 3

Now we consider a tight lower bound for the problem. Any algorithm that solves this problem can be used to count the number of inversions in a permutation π of the integers 1 through n. On the unit circle, for $1 \leqslant k \leqslant n$, label the point $\exp(i\pi((n-k+1)/2n))$ as k and the point $\exp(i\pi((k+n)/2n))$ as π_k. Figure 3 shows the construction for $\pi = 5\ 4\ 1\ 3\ 2$. We claim that the number of inversions in permutation π is equal to the number of intersections of the n chords drawn between identically labeled endpoints.

Consider chord i: the number of chords j with $j > i$ that cross chord i is the same as entry b_i in the inversion table for π defined in Section 5.1.1 of [1]. Hence the total number of intersections between chords is the same as the sum of the b_i's, which is just the number of inversions in π.

In a comparison-based model of computation, counting the inversions in a permutation of the integers 1 through n is known to require time $\Theta(n \log n)$ (Exercise 5.3.1-29 in [1]). The algorithm given above meets this time bound. The lower bound for other models of computation is not known.

1. Donald E. Knuth, "The Art of Computer Programming," Vol. 3, "Sorting and Searching," 2nd ed., Addison-Wesley, Reading, Mass., 1973.

[Also solved by V. Akman (Rensselaer Polytechnic Institute).]