Code Jam 2010 - Round 1C

Analysis: Rope Intranet

This problem was easier than it could have been, since the constraints don't require you to write an efficient solution. To solve it you can simply iterate through each pair of ropes and test if they intersect. Checking for intersection can be done in various ways. One way is to write the two line equations and then solve a system of linear equations with two variables to find the intersection point. An easier solution is to simply check if the order of the ends of the pair of ropes on the first building is the opposite of the order of the ends of the ropes on the second building. This translates to the code:

```
(A[i] - A[j]) * (B[i] - B[j]) < 0
```

This algorithm takes $O(n^2)$ and it's fast enough to solve our problem.

This problem is very similar to the classic one which asks for the number of inversions within a given permutation. An inversion for a permutation p is a pair of two indexes $i < j$ such that $p_i > p_j$. Let's see why these problems are equivalent. If *ra* is the rank of *A[i]* when we sort *A* and *rb* is the rank of *B[i]* when we sort *B*, then the ropes problem becomes the inversion count problem on the permutation p where $p_{ra} = rb$ for each i.

This new problem is a good application for divide and conquer algorithms, and can be solved in $O(n \log n)$ time. Merge sort can be adapted nicely to not only sort an array, but count the number of inversions as well. Other solutions use data structures like segment trees, augmented balanced binary search trees or augmented skip lists.

Number of inversions in a permutation