CS 577: Introduction to Algorithms — Homework 2

Out: 02/08/18 — **Due: 02/15/18**

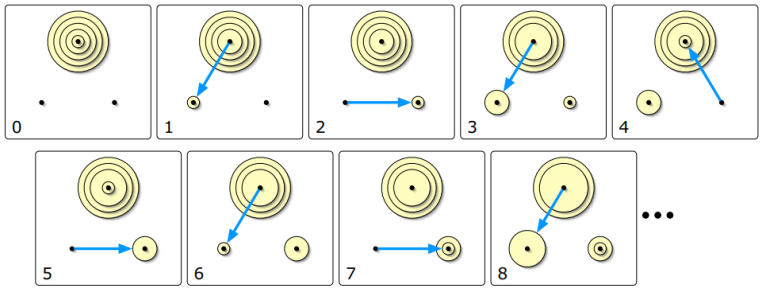
Ground Rules

- The homework is worth 5 points (out of a total of 100 points you can accumulate in this course).
- The homework is to be done and submitted in pairs. You can partner with someone from either section.
- The homework is due at the beginning of either lecture on the due date.
- No extensions to the due date will be given under any circumstances.
- Turn in your solution to each problem on a **separate sheet** of paper (or sheets stapled together), with your names clearly written on top.

Problems

1. **(Worth: 2 points. Page limit: 1 sheet; 2 sides)** Consider the following variant of the Tower of Hanoi puzzle. As in the original puzzle, you have three pegs, numbered 0, 1, and 2, and you have n disks on peg 0 that are to be moved to peg 2, subject to the constraint that only one disk can be moved at a time, from one peg to another, and that no disk is allowed to be on top of a smaller disk (this last constraint is also satisfied initially).

   Unlike the original puzzle, however, now the pegs are arranged in a circular fashion, as 0-1-2 in counterclockwise order, and you are only allowed to move disks counterclockwise. See figure.



A top view of the first eight moves in a counterclockwise Towers of Hanoi solution

   (a) Develop a divide and conquer algorithm for solving this problem. Although we do not explicitly require a proof of correctness for your algorithm, you will receive full points only if your algorithm satisfies all requirements for the problem.

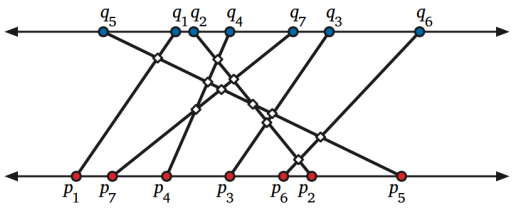
   Hint: Note that for this variant, moving a disk one place counterclockwise can be different from moving it two places counterclockwise. We recommend writing two different procedures for these two operations and linking them through recursive calls.

   (b) Analyze the asymptotic number of moves your algorithm makes. For full points, your algorithm should make $O(3^n)$ moves.

(*See next page for problem 2*)

2. **(Worth: 3 points. Page limit: 2 sheets; 4 sides)**

(a) Suppose you are given two sets of n points, one set $\{p_1, p_2, \cdots, p_n\}$ on the line $y = 0$ and the other set $\{q_1, q_2, \cdots, q_n\}$ on the line $y = 1$. Create a set of n line segments by connecting each point p_i to the corresponding point q_i. Your goal is to develop an algorithm to determine how many pairs of these line segments intersect. Your algorithm should take the $2n$ points as input, and return the number of intersections. It should run in $O(n \log n)$ time.



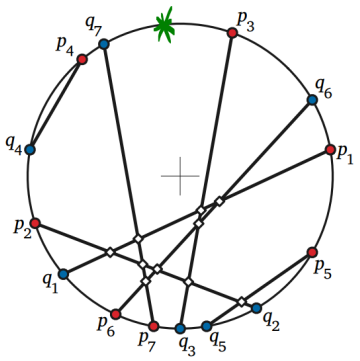
An example for the line intersection problem.

Describe a divide and conquer algorithm for this problem. Prove its correctness and analyze its running time.

Hint: What does this problem have in common with the problem of counting inversions in a list? Can you "reduce" (or "translate") this problem to the problem of counting inversions in an appropriately defined list?

(b) Now suppose you are given two sets $P = \{p_1, p_2, \cdots, p_n\}$ and $Q = \{q_1, q_2, \cdots, q_n\}$ of n points on the unit circle. Connect each point p_i to the corresponding point q_i. Once again our goal is to determine the number of intersections among the resulting line segments.

Note that the number of intersections do not depend on the actual positions of the points along the circle, but only on their position relative to other points, for example, for each i, which other pairs (p_j, q_j) have exactly one end-point in the arc clockwise from p_i to q_i. Therefore, for simplicity we will assume that the points are numbered and specified according to their appearance in clockwise order starting from some arbitrary location, call it "origin", on the circle. So, for example, if $p_1 = 3$, then p_1 is the third point clockwise from the origin in the set $P \cup Q$ (as in the figure below, where origin is denoted with a green asterisk).



Given the sets P and Q in this format, we will develop a divide and conquer algorithm to determine the number of intersections. Imagine splitting the circle into two parts, each of which contains half the points. What subproblems do these parts correspond to? Determine how to solve each of these subproblems, and put the components together in order to achieve an overall solution.

Prove the correctness of your algorithm and analyze its running time. Your algorithm should run in $O(n \log^2 n)$ time.

Hint: Use your solution to part (a) as a subroutine.