Due Date: November 3, 2015

Problem 1: Let P be a convex polygon in $\mathbb{R}^2$ with its sequence of vertices $\{p_1, \ldots, p_n\}$ in clockwise order, with p_1 being the leftmost vertex. Assume that this sequence of vertices is stored in an array.

(a) Describe an $O(\log n)$-time algorithm that determines whether a given point q lies inside P.

(b) Describe an $O(\log n)$-time algorithm to compute the tangents of a point q to P.

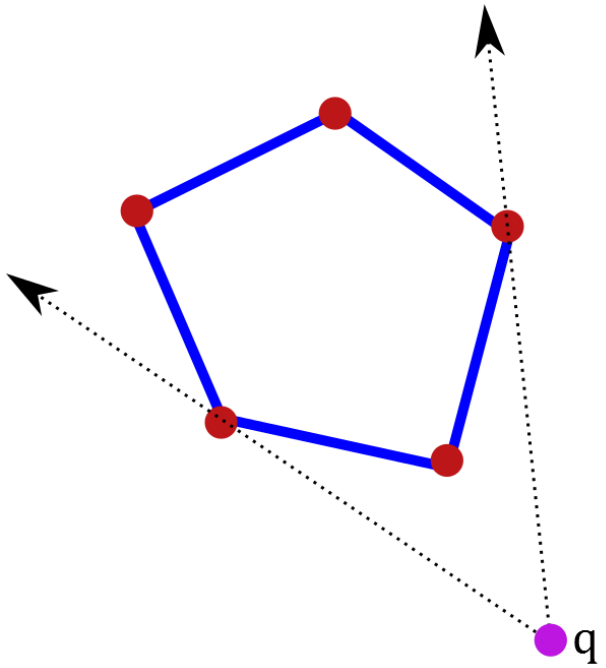

Figure 1: A convex polygon P and the two tangents from a point q to P.

(c) (**Extra Credit**) Let Q be another convex polygon in $\mathbb{R}^2$ with its sequence of vertices $\{q_1, \ldots, q_m\}$ in clockwise order, with q_1 being the leftmost vertex, that is disjoint from P. Describe an $O(n+m)$ algorithm to determine the outer tangents between P and Q.

Problem 2: In a word processor, the goal of "pretty-printing" is to take text with uneven lines and turn it into a text whose right margin is as "even" as possible. To make this precise, we need to define what it means for right margins to be "even." So suppose our text consists of a sequence of words $W = \langle w_1, w_2, \ldots, w_n \rangle$, where w_i consists of c_i characters. We have a maximum line length L. We assume we have a fixed-width font and ignore issues of punctuation or hyphenation.

A *formatting* of W consists of a partition of the words in W into lines. In the words assigned to a single line, there should be a space after each word except the last; and so if

$w_j, w_{j+1}, \ldots, w_k$ are assigned to one line, then we should have

$$\left[\sum_{i=j}^{k-1}(c_i + 1)\right] + c_k \leq L.$$

We will call an assignment of words to a line *valid* if it satisfies this inequality. The difference between the left hand side and right hand side will be called the *slack* of the line- that is, the number of spaces left at the right margin.

Give an efficient algorithm to find a partition of a set of words W into valid lines, so that the sum of the *squares* of the slacks of all lines (including the last line) is minimized.

Problem 3: Let $S = \{p_1, p_2, \ldots, p_n\}$ be a set of points, where $p_i = (x_i, y_i)$. Point p_i **dominates** point p_j if $x_i > x_j$ and $y_i \geq y_j$, or $x_i \geq x_j$ and $y_i > y_j$.

We say a point $p_i \in S$ is *maximal* if p_i is not dominated by any other point of S. If we draw horizontal and vertical lines through a maximal point, there will be no point in the upper-right quadrant. For example, see Figure 2.

Provide a $O(n \log n)$-time algorithm to compute the set of maximal points of S.

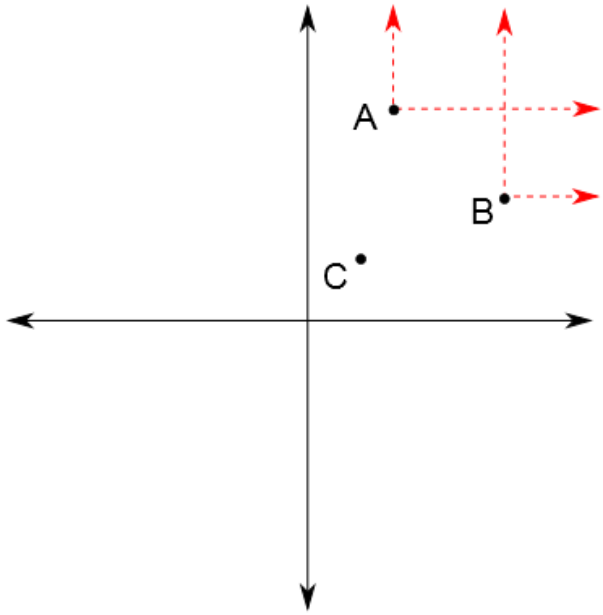

Figure 2: Points A and B both dominate point C, but do not dominate each other. Therefore the maximal set is $\{A, B\}$.

(**Extra credit:**) For points in $\mathbb{R}^3$, $p_i = (x_i, y_i, z_i)$, we say that p_i dominates p_j if the points are distinct ($p_i \neq p_j$) and the following inequalities hold: $x_i \geq x_j, y_i \geq y_j$, and $z_i \geq z_j$. Provide an $O(n \log n)$ algorithm to determine the set of maximal points of a set $S = \{p_1, p_2, \ldots, p_n\}$ when the points lie in $\mathbb{R}^3$.

Problem 4: Any string can be decomposed into a sequence of palindromes. For example the string `BUBBASEESABANNA` can be broken into palindromes in the following ways (and many others):

$$\texttt{BUB} \cdot \texttt{BASEESAB} \cdot \texttt{ANANA}$$
$$\texttt{B} \cdot \texttt{U} \cdot \texttt{BB} \cdot \texttt{A} \cdot \texttt{SEES} \cdot \texttt{ABA} \cdot \texttt{NAN} \cdot \texttt{A}$$

Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string `BUBBASEESABANNA`, your algorithm would return 3.

Problem 5: In the currency exchange market, a sequence of trades starting and ending at the same currency which results in more money than you started with is called an **opportunity cycle**. For example if the exchange rates were \$1 for 120 yen, 1 yen for 0.01 euros, and 1 euro for \$1.5, then by trading our money through this cycle once we could turn \$1 into \$1.80.

Suppose we have a $n \times n$ matrix E of exchange rates for n currencies, such that the entry $E[i, j]$ tells you how much of currency j can be obtained with one unit of currency i. (Do *not* assume that $E[i, j] \cdot E[j, i] = 1$.) Give an efficient algorithm which returns an opportunity cycle if one exists.