Instructions. Collaboration is permitted in groups of size at most three. You must write the names of your collaborators on your solutions and must write your solutions alone.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

1. **John’s Ellipse in 2D**

There is a theorem of John’s that states that given a polytope \( P \) there is a unique maximum volume ellipsoid contained in \( P \). These John’s ellipsoids are now the basis for the fastest and latest LP solvers. Here we hope to understand these ellipsoids better by starting with the 2D case. In particular, the goal of this problem is to find this ellipse in 2D in linear time.

Suppose the input to our problem is \( n \) halfspaces in 2D, \( H = \{h_1, \ldots, h_n\} \) in general position. We would like to find the John’s ellipse contained in \( \cap H \). We will assume the feasible region is nonempty. Note that we need three constraints to ensure that the ellipse is bounded, the feasible region is a triangle. In this case the ellipse is known as the Steiner inellipse, see [https://en.wikipedia.org/wiki/Steiner_inellipse](https://en.wikipedia.org/wiki/Steiner_inellipse). You should convince yourself that given a triangle you can compute the Steiner inellipse and test if a line intersect the ellipse all in constant time.

1. Give a linear time algorithm which given \( H \) determines if \( H \) is bounded and if so finds three half spaces in \( H \) whose intersection is bounded.

2. Give a random incremental algorithm that given \( H \) as input finds the John’s ellipse in expected linear time.

Hint: Consider the simpler problem of finding the John’s ellipse given that you have a guarantee that the boundaries \( \partial h_1 \) and \( \partial h_2 \) touch the John’s ellipse of \( H \).

2. **Maximum Slope**

Suppose we have \( n \) red points \( p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n) \) such that \( x_i > 0 \) for \( i \in [1, n] \) and we have \( n \) green points \( q_1 = (x'_1, y'_1), \ldots, q_n = (x'_n, y'_n) \) such that \( x'_i < 0 \) for
\( i \in [1, n] \). The goal is to find a pair \((q_i, p_j)\) with maximum slope, that is it maximizes \((y_j - y'_i)/(x_j - x'_i)\).

Give an expected linear time algorithm to find the pair with maximum slope.

3. Densest Subgraph

In this problem, we are going to design an algorithm that calculates the densest subgraph for any undirected graph \(G\) where the density of any graph \(H\) is defined by the ratio of the number of edges in \(H\) to the number of vertices in \(H\). As preparation, we will build up some tools by solving some simpler problems.

(a) (15) Consider a directed graph \(G\) where each vertex \(v\) is labeled with value \(p_v\) which may be positive or non-positive. The task is to find the closure subgraph \(H\) of \(G\) with maximum total value, where a subgraph \(H\) is a closure if for each vertex \(v \in H\), all its successors in \(G\) are in \(H\) as well.

(b) (15) Consider an undirected graph \(G\) where each vertex \(v\) is labeled with value \(p_v\) and each edge \(e\) is labeled with value \(c_e\). The task is to find the subgraph \(H\) of \(G\) with maximum total value among vertices and edges. Use the algorithm in part (a) as a black box to solve this problem.

(c) (10) Design an algorithm that calculates the densest subgraph for a given undirected graph \(G\). Use the algorithm in part (c) as a black box to solve this problem.