Prob. 1	Prob. 2	Prob. 3

Problem 1.

Let's begin with the main intuition of the algorithm before expliciting it. For each edge of our polygon, the inside of our simple polygon lies only on one side of this edge. If there exists a point within the polygon from which it is possible to draw a segment to every vertex of the polygon without crossing the boundary of the polygon, then this point must be on the inside-side of each edge. Let's formulate this as a property and then prove it.

Proposition
There exists a point within a simple polygon from which it is possible to draw a segment to every vertex of the polygon without crossing the boundary of the polygon **iff** the intersection of the half-planes defined as the inside-side of each and every edge of the polygon is non-empty.

Let's prove it. First, let's assume there exists such a point P. The line between this point and a given vertex is in the inside of the polygon, then this point is the inside-side of the two edges associated to this vertex. Therefore, this point P has to be on the inside-side of every edge of the polygon.
Reciprocally, let's assume the intersection of the half-planes defined as the inside-side of each and every edge of the polygon is non-empty. Let's consider a point P in this intersection. We need to show that the whole line between this point and any vertex is fully inside the polygon. Let's consider our line crosses an edge. As our line is going to be inside at its beginning (near P) and at its end (it reaches a vertex by the inside-side of the two edges defining this vertex), it needs to cross at least another edge to re-enter the polygon. This means that our line goes from the outside to the inside of this edge. Therefore P is on the outside-side of this edge which is impossible. Thus, the whole line between P and any vertex is fully inside the polygon.

Now that we have found a necessary and sufficient condition for our problem, we can describe an algorithm to solve it using the randomized incremental construction algorithm we saw in class to solve 2D linear optimisation problems. Our algorithm is the following :

- We first need to find where is the interior of the polygon given the list of the edges. This is easily done in linear time, for instance by suming the angles knowing that the sum of the interior angles is less than the sum of the exterior ones.
- Then we transform our problem into an optimization one, considering that beeing on the interior-side of edge i is giving us an inequality of the form $a_{i,1}x + a_{i,2}y \leqslant b_i$. Now we choose a function to optimize, we can take whatever we want so let's pick $f(x,y) = x$. This transformation can be done in time linear to the number of edges.

- Finally, we solve this problem with the 2D linear optimisation algorithm seen in class. This algorithm runs in expected linear time. The existence of a solution to the optimisation problem is equivalent to the existence of a point within the polygon from which it is possible to draw a segment to every vertex of the polygon without crossing the boundary of the polygon.

The three steps of our algorithm run in linear time or expected linear time. Therefore our algorithm runs in expected linear time.

Problem 2.

Just like for the question 1, we will transform this problem in a 2D linear optimization problem, so that we can use the (expected) linear runing-time algorithm already studied during the course.

Let suppose that we have n red points, of coordinates $(rx_i, ry_i) \forall i \in 1,..,n$ and m green points of coordinates $(gx_i, gy_i) \forall i \in 1,..,m$. Our goal is to find a line separating the green points from the red points (we suppose that our computer is not affected by daltonism). Such a line has a Cartesian equation of the form $y = n_1 x + n_2$. We want to find n_1 and n_2 (the parameters of the line).

A point is said "under" a line of parameters (n_1, n_2) if its coordinates (x, y) verify $y \leq n_1 x + n_2$, while it is considered "above" the line if we have $y \geq n_1 x + n_2$.
This can be rewritten as $(-x)n_1 + (-1)n_2 \leq -y$ for "under", and $xn_1 + n_2 \leq y$ for "above". Those are constraints for a 2D linear optimization problem.
Thus, we will specify our constraints like $(-rx_i)n_1 + (-1)n_2 \leq -ry_i \forall i \in 1,..,n$ for red points and $gx_i n_1 + n_2 \leq gy_i \forall i \in 1,..,m$ for green points.
We can use nearly whatever we want as the function to optimize, like maximizing $f(n_1, n_2) = 1$ (really simple function to optimize) or $f(n_1, n_2) = \frac{1}{1 - n_1^2 + n_2^2}$ (so that we get normalized parameters).

Thanks to this reformulation of the problem we can easily claim that we have a $O(n+m)$ expected running time algorithm solving that problem: we use the algorithm as seen during the class where we incrementally find a solution satisfying more and more constraints.

Problem 3.