Problem 1

Part A

Prove that the intersection of two convex sets is again convex. This implies that the intersection of a finite family of convex sets is convex as well.

Proof. Let A and B be convex sets. Take $p, q \in A \cap B$. Note that both p, q must then be in A and B, and since both are convex it follows $\overline{pq} \in A$ and $\overline{pq} \in B$. But this just means $\overline{pq} \in A \cap B$. $\diamond$

Part B

Prove that the smallest perimeter polygon $\mathcal{P}$ containing a set of points P is convex.

Proof. Assume towards contradiction that $\mathcal{P}$ is the smallest perimeter polygon containing P but is not convex. Then there must exist two points in $\mathcal{P}$ with their line L not contained in $\mathcal{P}$. We can pick $p, q \in L$ such that p, q are on the boundary of $\mathcal{P}$ since L must intersect the boundary in at least 2 spots. Note that the boundary path connecting p and q cannot be a straight line, otherwise non-convexity would be violated. Therefore the line $\overline{pq}$ provides a strict lower bound on the distance between p and q compared to all other paths, including the current boundary. By replacing the boundary path between p and q with the line $\overline{pq}$, the overall perimeter of $\mathcal{P}$ would decrease. However, this means $\mathcal{P}$ could not have been the minimum perimeter polygon, a contradiction. $\diamond$

Part C

Prove that any convex set containing the set of points P contains the smallest perimeter polygon $\mathcal{P}$.

Proof. Assume towards contradiction that A is a convex set containing P that does not contain $\mathcal{P}$. Consider $A \cap \mathcal{P}$. Since A does not contain $\mathcal{P}$, there must be a portion of $\mathcal{P}$ outside of A. This portion can be replaced by the corresponding portion of $A \cap \mathcal{P}$. Using (a) and (b) we know that $A \cap \mathcal{P}$ must be a convex polygon. Since it is an intersection of polygons, the perimeter must also be smaller than $\mathcal{P}$, a contradiction. $\diamond$

Problem 2

> Describe an $O(n \log n)$ time method for determining if two sets, A and B, of n points in the plane can be separated by a line.

The main idea for the following algorithm is that two disjoint convex sets always have a line that separates them (hyperplane separation theorem). Therefore if A and B have convex hulls that are disjoint, there must be a line that can separate the hulls and therefore the original sets of points. Since the computed convex hulls return just the boundary, testing for the set intersection requires checking

1. None of the edges intersect
2. Neither convex hull is contained within the other

If it is determined that none of the edges intersect, then the convex hulls can only be in 2 configurations: one is contained entirely inside the other, or they are separated. This means that checking if a convex hull is contained in another simplifies to checking if any point of one is contained in the other. In total then, the algorithm is ($\star$)

Algorithm 1 Determine if two point sets A and B are separable

```
1:  procedure CanSeperate(A, B)
2:      H1 ← ConvexHull(A)
3:      H2 ← ConvexHull(B)
4:      if EdgesIntersect(H1, H2) then
5:          return False
6:      else if H1 is contained in H2 then
7:          return False
8:      else if H2 is contained in H1 then
9:          return False
10:     else
11:         return True
12:     end if
13: end procedure
```

The time complexity can be broken down by examining each major line of the algorithm.

2 and 3) Finding the convex hull per set of points takes $O(n \log n)$ using the Graham Scan algorithm.

4) Determining if any of the edges intersect can be done via a sweeping line algorithm in $O(e \log e)$ where e is the total number of edges in both $\mathcal{H}_1$ and $\mathcal{H}_2$. Since the number of edges per convex hull is $O(n)$, EdgesIntersect is $O(n \log n)$.

6 and 8) As previously discussed (see ($\star$) above), checking if either convex hull contains the other can be done by just picking any point from one set and then checking if it is contained in the other. This can be solved multiple ways, but an example is using the winding number which can be found in a single pass over the edges. Therefore the time complexity is just $O(n)$.

The total time complexity then is $O(2n \log n + n \log n + 2n) = O(n \log n)$.

Problem 3

Part A

Show that the sign of the determinant

$$D = \begin{vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{vmatrix}$$

determines whether r lies left or right of the line.

Proof. First note that

$$\begin{aligned} D &= \begin{vmatrix} q_x & q_y \\ r_x & r_y \end{vmatrix} + \begin{vmatrix} p_x & p_y \\ r_x & r_y \end{vmatrix} + \begin{vmatrix} p_x & p_y \\ q_x & q_y \end{vmatrix} \\ &= (q_x r_y - r_x q_y) + (p_x r_y - r_x p_y) + (p_x q_y - q_x p_y). \end{aligned}$$

If we let u be the vector from p to q and v be the vector from q to r, then

$$u = \begin{bmatrix} q_x - p_x \\ q_y - p_y \\ 0 \end{bmatrix}, v = \begin{bmatrix} r_x - q_x \\ r_y - q_y \\ 0 \end{bmatrix}$$

and

$$\begin{aligned} u \times v &= \begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ q_x - p_x & q_y - p_y & 0 \\ r_x - q_x & r_y - q_y & 0 \end{bmatrix} \\ &= \hat{k}\left[(q_x - p_x)(r_y - q_y) - (r_x - q_x)(q_y - p_y)\right] \quad \text{(Expand on col. 3)} \\ &= \hat{k}\left(q_x r_y - r_x q_y + p_x r_y - r_x p_y + p_x q_y - q_x p_y\right) \\ &= D\hat{k}. \end{aligned}$$

When discussing the cross product, the direction in which the vector points is determined by the right hand rule. Since u and v lie in the xy-plane, then the direction of the cross product is exactly the sign of the

$\hat{k}$ component which is D. Applying the right hand rule implies that D is positive when r is on the left of $\overline{pq}$ and to the right when D is negative. ◇

Part B

Show that $|D|$ in fact is twice the surface of the triangle determined by p, q, and r.

Proof. The cross product $u \times v$ has the property that its magnitude equals the area of the parallelogram where its two sides are u and v. Note that the choices of u and v from (a) are sides of the triangle $\triangle pqr$. Since the parallelogram is two copies of this triangle and the magnitude of the cross product is the area of this parallelogram, $\sqrt{0^2 + 0^2 + D^2} = |D|$ would equal twice the area of the triangle $\triangle pqr$. ◇

Part C

Why is this an attractive way to implement the basic test in algorithm ConvexHull? Give an argument for both integer and floating point coordinates.

1. For either integer or floating point, the calculation is a fixed number of math operations meaning it can be done in constant time.
2. There is no need for angles or trig functions which would have required integer coordinates to be converted to floating point and could introduce potential numerical inaccuracies/be less robust
3. With integer arithmetic, the expression is well defined and **exact** as long as it doesn't overflow or underflows
4. Multiplication and addition are simple enough that it is possible to leverage arbitrary precision floating point numbers without incurring a high overhead cost

Problem 4

Let S be a set of n line segments in the plane. Prove that the convex hull of S is exactly the same as the convex hull of the $2n$ endpoints of the segments.

Proof. Let $\mathcal{H}_s$ be the convex hull of the the line segments S and $\mathcal{H}_p$ the convex hull of the endpoints of the segments of S. For a line segment $L \in S$, it has endpoints p and q. Since $\mathcal{H}_p$ must contain these points and is convex, then L must be in $\mathcal{H}_p$. Hence $\mathcal{H}_s \subseteq \mathcal{H}_p$. Note that any endpoint p must be contained in a line segment from S, meaning $\mathcal{H}_p \subseteq \mathcal{H}_s$. Since both convex hulls are subsets of each other, they must be equal. $\diamond$

Problem 5

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two disjoint convex polygons with n vertices in total. Give an $O(n)$ time algorithm that computes the convex hull of $\mathcal{P}_1 \cup \mathcal{P}_2$.

The key idea is that two convex polygon have two tangent lines, an upper and lower, that can be used to connect them together to be a larger convex polygon. Therefore the following algorithm finds the two tangents and then creates the new polygon by walking around the polygons starting at the end of the upper tangent and going around, across the lower tangent, and then around until the starting point of the upper tangent adding the points along the way.

Algorithm 2 Merge two convex polygons into their convex hull

Input: Convex polygons $\mathcal{P}_1$ and $\mathcal{P}_2$ with vertices in clockwise order

Output: The convex hull $\mathcal{H}$ of $\mathcal{P}_1$ and $\mathcal{P}_2$ in clockwise order.

procedure ConvexUnion($\mathcal{P}_1, \mathcal{P}_2$)
    $\mathcal{H} \leftarrow$ Empty point list
    $T_{\text{upper}} \leftarrow$ UpperTangent($\mathcal{P}_1, \mathcal{P}_2$)
    $T_{\text{lower}} \leftarrow$ LowerTangent($\mathcal{P}_1, \mathcal{P}_2$)
    Append the points of $\mathcal{P}_2$ to $\mathcal{H}$ starting from the endpoint of T_{upper} and going clockwise until and including the starting point of T_{lower}
    Append the points of $\mathcal{P}_1$ to $\mathcal{H}$ starting from the endpoint of T_{lower} and going clockwise until and including the starting point of T_{upper}
    return $\mathcal{H}$
end procedure

The upper tangent algorithm is as follows. The lower one is essentially identical except the turns considered just need to be reversed when checking lower tangency.

Algorithm 3 Find the upper tangent starting at $\mathcal{P}_1$ and ending at $\mathcal{P}_2$

procedure UpperTangent($\mathcal{P}_1, \mathcal{P}_2$)
    $v_i \leftarrow$ vertex of $\mathcal{P}_1$
    $q_j \leftarrow$ vertex of $\mathcal{P}_2$
    while $\overline{p_i q_i}$ is not an upper tangent **do**
        while $p_i q_j$ is not tangent to $\mathcal{P}_1$ **do**
            $p_i \leftarrow p_{i-1}$ ▷ Move counter clockwise on $\mathcal{P}_1$
        end while
        while $p_i q_i$ is not tangent to $\mathcal{P}_2$ **do**
            $q_j \leftarrow q_{j+1}$ ▷ Move clockwise on $\mathcal{P}_2$
        end while
    end while
end procedure

Breaking down the time complexity:

1. Checking if a line forms a tangent to a convex polygon can be done in constant time. In the context of the UpperTangent algorithm, this can be done by considering the turns $p_i \rightarrow q_j \rightarrow q_{j+1}$ and $p_i \rightarrow q_j \rightarrow q_{j-1}$ and seeing if they are the same kind of turn. If they are not, then the line cannot be tangent as it would therefore have two points on opposite sides of it and must be going through the polygon. Checking the turn directions is constant time so checking tangency is constant time.

2. For the while loop that checks for the upper tangent, the worst case scenario is that all points of $\mathcal{P}_1$ and $\mathcal{P}_2$ must be cycled through each time until the upper tangent is found. Since we are guaranteed that an upper tangent does exist and can bound the number of iterations, this worst case does terminate. Therefore UpperTangent and LowerTangent are $O(n)$.

3. For ConvexUnion, the appending process only visits a given vertex at most 1 time. Since there is at most n vertices, it is bounded by n and is $O(n)$.

In total then, the call to find both tangents and then walk the polygons and tangents to produce the final hull takes $O(n + n + n) = O(n)$