Problem 1. Let $A$ and $B$ be sequences of points in the Cartesian plane such that the elements of $A$ (resp., $B$) are the coordinates of the corners of a convex polygon $A$ (resp., $B$).

That is, if $|A| = k$, the convex polygon is such that there is an edge connecting $A[i]$ to $A[(i + 1) \mod k]$, for $0 \leq i \leq k - 1$. The same holds for $B$.

Design an algorithm that given $A$ and $B$ constructs a sequence $C$ such that the elements of $C$ are the coordinates of the corners of a convex polynomial $C$ which is a convex hull for the set of points given by the union of the corners of $A$ and $B$. Your algorithm should run in time $O(|A| + |B|)$.

Describe your algorithm, a simplified analysis of its correctness, and analyze its running time.

Solution. See page 14 of these lecture notes from CMU:

[https://www.cs.cmu.edu/afs/cs/academic/class/15456-s14/Handouts/cmse754-lects.pdf](https://www.cs.cmu.edu/afs/cs/academic/class/15456-s14/Handouts/cmse754-lects.pdf)

The general idea is to compute two tangents to the two convex hulls. Given these two tangents, merging the convex hulls is done by taking the “left” portion of the leftmost hull and the “right” portion of the rightmost hull.
Problem 2. In the fuzzy pattern-matching problem given a text $T$ and a pattern $P$, we are looking to verify if the pattern $P$ appears in the text $T$. However, we are accepting as valid matches occurrences of $P$ in $T$ for which, at most, one character is mismatched. E.g, let $T = aaaaaabbd$ and $P = abc$, there is a fuzzy match of $P$ in $T$ for $abd$ with at most one mismatch.

1. Show how to modify the Rabin-Karp algorithm for this problem. Beside the initialization phase, the algorithm should run in the expected time $O(n)$.

2. Let $\Sigma$ denote the set of possible symbols. What is the expected running time of the algorithm when including the initialization?

Solution.

1. The idea here is to make use of the fact that we know our pattern at initialization time. Therefore, we can compute every possible fuzzy match and store these hashes in a set (or some other expected constant-time lookup data structure). When we need to search for a match in a text $T$, we can simply run a modified version of Rabin-Karp where instead of looking for hash equality, we check to see if the hash of the current substring is in the set of pre-computed hashes. Since looking up the hash can be done in constant time, the asymptotic expected runtime is the same as Rabin-Karp: $O(|T| + |P|)$.

2. The total number of fuzzy-matches we need to pre-compute is given by $|\Sigma||P|$. Therefore, the overall runtime including initialization is $O(|\Sigma||P| + |T|)$.

\[\square\]
Problem 3. Hammad is mixing paint for his house. Since his favorite color is green he’s mixing some blue paint and yellow paint together with some paint thinner. He’s managed to create $n$ different shades of green with all the blue and yellow paint that he has.

For example, suppose he made three different shades of green.

<table>
<thead>
<tr>
<th>Compound</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yellow</td>
<td>0.7</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>Blue</td>
<td>0.2</td>
<td>0.1</td>
<td>0.7</td>
</tr>
<tr>
<td>Paint Thinner</td>
<td>0.1</td>
<td>0.6</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Then it is possible to produce a shade of green that is 35% yellow and 27.5% blue by mixing the shades he currently has in a $1 : 2 : 1$ ratio ($25\%$ $S_1$, $50\%$ $S_2$, $25\%$ $S_3$). However, it is impossible to create the shade of green which is 20% yellow and 10% blue.

Design an $O(n \log n)$ algorithm that checks whether it’s possible to create a liquid with the specified percentage of yellow and blue. Argue the correctness of your algorithm.

Example Input: $[(0.7, 0.2), (0.3, 0.1), (0.1, 0.7)]$, (0.35, 0.275)

Output: True

Hint: What is this nonsense about paint colors? I wonder! I guess possible ratios of Yellow and Blue used to obtain the $n$ shades of green look like coordinates of points on the plane... :)

Solution. Algorithm

Algorithm 1 Paint Mixing

```python
procedure PointInclusion(P, q)
    xMax ← $p.x \in P$ such that $p.x$ is maximal
    horizontalSegment ← $(q, (xMax, q.y))$
    $E \leftarrow \{\}$
    for $i$ in range(0, $S.size()$) do
        $j \leftarrow (i+1) \mod S.size()$
        $E.append((S[i], S[j]))$
        intersectingEdges ← $E.filter(e \Rightarrow$ INTERSECT(horizontalSegment, $e$))
    return intersectingEdges.size().isOdd()

procedure Mixable(S, l)
    S.sort(OrientationComparator)
    GrahamScan(S)
    return PointInclusion(S, l)
```

Correctness: The algorithm checks to see if a certain shade of paint is mixable by checking to see if it is contained in the convex hull formed by the original shades of paint. Let $C = \{v_1, \ldots, v_m\}$ denote the convex hull of the original shades of paint. Notice that a shade is only mixable if and
only if it can be represented as a linear combination \( a_1v_1 + \cdots + a_mv_m \) where \( a_1 + \cdots + a_m = 1 \), i.e. a convex combination of \( v_1, \cdots, v_m \). One can think of each \( a_i \) as the percentage of the shade \( v_i \) used to create the mixed shade. By the hint, the convex hull is the set of all convex combinations of the original \( n \) shades and thus every mixable paint is an element of the convex hull and every point in the convex hull represents a mixable paint.

**Runtime:** We begin by preforming an \( O(n \log n) \) sort on the original set of paints. We then preform the Graham scan on this set of paints to form the convex hull, which is done in \( O(n) \) time. Finally, we check to see if the input shade is contained within the convex hull using point inclusion. Determining the maximum \( x \) coordinate in our convex hull requires a linear scan of the hull which in the worst case contains \( O(n) \) points. We can create the horizontal segment for the point we are checking in constant time and extract all the edges from our convex hull in linear time. Finally, filtering the set of edges to determine the number of intersections with our horizontal line segment also requires \( O(n) \) time. In total checking if a point is included in the convex hull requires at worst \( O(n) \) operations. Thus, the overall algorithm requires \( O(n \log n) \) time where we are limited by time required to preform our original sort.