Homework 8: Pattern Matching & Computational Geometry

Problem 1. In the fuzzy pattern-matching problem given a text T and a pattern P constructed using symbols from an alphabet Σ, 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 $T = aaaaaabd$ 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:

   - The initialization should require almost at most $c_1|\Sigma||P|$ time where c is a constant value with respect to $|P|$, T, and $|\Sigma|$.
   - *Besides for the initialization phase*, the algorithm should run in the expected time $c|T|$, where c is a constant values with respect to $|P|$, T, and $|\Sigma|$. You can assume a constant number of collisions with high probability.

   Overall, the expected running time of your algorithm should be $\mathcal{O}(|\Sigma||P| + |T|)$. NOTE: You cannot assume $|\Sigma|$ to be constant with respect to $|P|$ and $|T|$.

2. Prove the correctness of your algorithm.

3. Analyze the worst-case running time of your algorithm

Solution. 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.

Therefore, the initialization phase is $O(|\Sigma||P|)$, since for each mismatch there are $|\Sigma|$ choices. Looking up the hash can be done in constant time, so the expected runtime post initialization is the same as Rabin-Karp: $O(|T|)$. This gives the overall expected runtime is $O(|\Sigma||P| + |T|)$. The worst case scenario is when we run into collisons, resulting in brute force checks. The runtime for this is $O(|\Sigma||P| + |T||P|)$. □

Problem 2. Lorenzo 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.

		Samples		
% Compound		S_1	S_2	S_3
	Yellow	0.7	0.3	0.1
	Blue	0.2	0.1	0.7
	Paint Thinner	0.1	0.6	0.2

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

```
procedure PointInclusion(P, q)
    xMax ← p.x ∈ P such that p.x is maximal
    horizonalSegment ← (q, (xMax, q.y))
    E ← {}
    for i in range(0, S.size()) do
        j ← (i + 1) mod S.size()
        E.append((S[i], S[j]))
    intersectingEdges ← E.filter(e => 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 $C = \{v_1, \cdots, v_m\}$ 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. □

Problem 3. Aditya is building a fence for his farm to protect against the invading squirrels. He plans on using fence posts with positions given by $Q = [q_0, q_1, \ldots, q_{n-1}]$, which are sorted in counterclockwise order with respect to q_0. Every fence post should be used as part of the resulting fence or contained entirely within it. To ensure he could always go to and from any place on his farm at the minimum distance possible, he constructed his fence in the shape of a convex polygon.

A year after building the fence, Aditya wants to add in a new fence post with position p. Describe a linear time algorithm that Aditya can use to modify his fence to include p (either as part of the fence or contained entirely within it) so that it still contains all the fence posts in Q. Show that your proposed algorithm is correct and analyze its running time.

Solution. First, use the standard point inclusion algorithm to check if p is contained in the current convex hull: if this is the case, we are done. If not, determine where p is in the counterclockwise order with respect to q_0 by iterating through Q and checking orientation. Now that we have the sorted order, we can apply the Graham Scan to get the new convex hull.

The runtime of this algorithm is $O(n)$: applying point inclusion is $O(n)$, checking where p is in the counterclockwise is just a linear time loop through the list, and the Graham Scan is $O(n)$. □