1. **Simple Paths and Convex Hull**

Suppose that $P = \{p_1, \ldots, p_n\}$ is a set of points in the plane. We say the the sequences of distinct points $Path = (p_1, \ldots, p_k)$ is a simple path if the line segments $l_i = [p_ip_{i+1}]$ are disjoint except for $l_i \cap l_{i+1} = p_{i+1}$. We may also allow $p_1 = p_k$ and in this case $l_{k-1} \cap l_1 = p_k$.

In the following questions we shall investigate the relation between finding a simple path of a set of points and finding their convex hull.

1. Design an algorithm for finding a simple path through all points in $P$. Make your algorithm as time efficient as possible.

2. In class we showed that computing the convex hull of $n$ points in a comparison based model requires $\Omega(n \log n)$ time. Show that given a simple path for these points one can find the convex hull in $O(n)$ time.

**HINT:**

The idea is to run a variant of incremental convex hull where we add the points in the order they appear on the path. Suppose we are given a simple path $Path = (p_1, \ldots, p_n)$ on $n$ distinct points and for simplicity no three are collinear. We start by constructing the triangle from the first three points and storing it as a doubly linked list of edges and recording which vertex is connected to the remain points on the path.

Let $I = \{i \mid p_i \in CH(p_1, \ldots, p_i)\}$ We will for each $i \in I$ incrementally compute the convex hull of $(p_1, \ldots, p_i)$. Make sure your algorithm handles the case when the point $p_{i+1}$ is interior to $CH(p_1, \ldots, p_i)$.

Use amortized analysis to show that your algorithm runs in $O(n)$ time.
3. Show that in general any comparison based algorithm that finds a simple path of the points in \( P \) requires \( \Omega(n \log n) \) comparisons.

(25) 2. Interval Trees
In the line segment intersection problem we used a BST to store information about the live line segments as we moved the sweep-line. For many problems a simpler data structure suffices, an **interval tree**.

Suppose that there is a set of intervals on the real line to be processed each with a left and right endpoint an integer between zero and \( n \). Thus we have \( n \) subintervals \([i, i + 1]\) for \( 0 \leq i < n \). Let \( T \) be a balanced binary tree with the each subinterval as a leaf in order. Now each internal node \( x \) of \( T \) represents the interval consisting of the subintervals contained in its subtree, denoted by \( I(x) \). We will let the follow set of nodes in \( T \) represent an interval \( I = [i, j] \):

\[
\text{Rep}(I) = \{ x \in T \mid I(x) \subset I \text{ and } I(P(x)) \not\subset I \} \quad \text{where } P(x) \text{ is the parent of } x.
\]

**Definition 0.1** If \( X \) is a subset of nodes of a tree \( T \) then we let \( \text{Tree}(X) \) denote be the smallest subtree of \( T \) containing \( X \) and closed under taking parent. The parent of the root is itself.

1. Show that \( \text{Rep}(I) \) contains at most \( 2 \log n \) nodes of \( T \) for any interval \( I \).

   Hint: How many nodes in a representation can be on a given level of the tree?

2. Show that the subtree \( \text{Tree}(\text{Rep}(I)) \) has at most \( O(\log n) \) nodes of \( T \).

3. Consider the following requests:
   - INSERT(I)
   - DELETE(I)
   - Query(i): Is \([i, i + 1]\) covered? i.e. Does \([i, i + 1]\) belong to some live interval.

   We say that an interval is **live** if it has been inserted but not deleted.

   Describe a data structure that handles these three updates in \( O(\log n) \) time per request. What attribute did you need to store at each node?

4. Consider a more general query:
   - Query(i, j): Is \([i, j]\) covered? i.e. Does every subinterval from \( i \) to \( j \) belong to some live interval?

   Describe a data structure that handles these four updates in \( O(\log n) \) time per request. What additional attributes did you need to store at each node?

5. Consider yet another query:
   - REPORT(i, j): Report the number of covered subintervals in the interval \([i, j]\).

   Describe a data structure that handles INSERT, DELETE, and REPORT in \( O(\log n) \) time per request.
3. **Circular Partition**

Given a set of red points $R$ and a set of green point $G$ in the plane give an algorithm to find a disk $D$ such that $G \subset D$ and $R \cap D = \emptyset$ if one exists. Your algorithm should run in expected linear time in the size of $R$ and $G$.

4. **Star Shaped Polygon**

A polygon $P$ is **star shaped** if there exists a point in the interior of $P$ that can see all of the interior.

1. Give an $O(n)$ expected time algorithm to determine if a simple polygon of size $n$ is star shaped.
2. Give a $O(\log n)$ time algorithm for determining if a point $q$ is in a star shaped polygon $P$. We assume that the vertices of $P$ are given in CW order and that we are also given a point $p$ that can see all of the interior of $P$. 
