1. Computing a Bounding Box

In this problem we discuss in more detail how to determine if an LP, Maximize $c^T x$ subject to $Ax \leq b$, is unbounded and return a proof that it is. If it is bounded we should return a subset $d$ constraints such that the LP on these constraints is still bounded.

- Lets start with the case $d = 3$. Suppose we have the three half spaces $\{h_1, h_2, h_3\}$ with normals $\{p_1, p_2, p_3\}$. Under what condition between the normals and $c$ will it be the case that our system is bounded, hint: think in terms of convexity and convex combinations.

- If someone gave you the $d$ constraints, Say $Ax \leq b$, how could they convince you that the LP is in fact bounded. If you had to find your own proof what would you do?

2. 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 $l_i = [p_i p_{i+1}]$ 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) 3. 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 $I = [i, j]$:

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

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

1. Show that $Rep(I)$ 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 $Tree(Rep(I))$ 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.

(25) **Star Shaped Polygon**

A polygon $\mathcal{P}$ is **star shaped** if there exists a point in the interior of $\mathcal{P}$ 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 $\mathcal{P}$. We assume that the 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 $P$.

(25) **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 $G$.

(20) **Broken Simple Path and Convex Hull Algorithm**

Suppose $P = \{p_1, ..., p_n\}$ is a set of points in the plane. We say that sequences of distinct points $\{p_1, ..., p_n\}$ form a simple path if the line segments $l_i = [p_i, p_{i+1}]$ are disjoint except for $l_i \cap l_{i+1}$. Furthermore we allow $p_1 = p_n$ and $l_{n-1} \cup l_1 = p_1$.

Given a simple path, $P$, consider the following algorithm to compute $CH(P)$. Let $S$ be a stack, initialized with $s_0, s_1$ where $s_0$ is the leftmost point of $P$ and $s_1$ is the clockwise successor in $P$. 

Page 3
1. While the top of the stack is not \(s_0\), take the next point in \(P\), some \(p_i\) along with the top two points in the stack, \(s_{t-1}, s_t\).

2. If \(s_{t-1}s_tp_i\) form a right turn, add \(p_i\) to the stack and continue.

3. While \(s_{t-1}s_tp_i\) form a left turn, pop the stack. Then add \(p_i\).

Clearly, this algorithm terminates. What’s more, it runs in \(O(|P|)\)! However, there’s a bug in this algorithm - find a counter-example for which this algorithm fails to find the convex hull of \(P\). Try to give a general description for the types of polygons this algorithm fails on.

(30) 7. * Area Of Rectangles*

Suppose that \(R_1, \ldots, R_n\) is a set of \(n\) axis aligned rectangles in the plane. The goal of this problem is to construct an \(O(n \log n)\) time algorithm to find the area of their union. There is several important and interesting ideas needed to get such a fast algorithm. The main algorithm design technique will be the sweep-line technique. But rather than use a BST to store out state we will use a much simpler data structure, the interval tree see Wikipedia. We assume that all the \(2n\) vertical(horizontal) sides have distinct values.

**Interval Trees**

(a) Show that our \(2n\) vertical sides will break the plan up into \(2n - 1\) strips.

(b) Explain how each interval can be represented by \(O(\log n)\) nodes in the interval tree where each leaf is a strip.

(c) Give an relatively simple sweep-line algorithm that runs in \(O(n^2)\) time and \(O(n)\) space.

2. More than likely your algorithm from part 1 had \(O(n)\) events and required \(\Omega(n)\) time per event. We shall say that a rectangle is live it intersects our sweep-line. Suppose that we have \(m\) live rectangles. These can be viewed as \(m\) segments. These \(2m\) endpoints will partition the line into \(2m + 1\) intervals.

Describe a binary search tree to represent these \(2m + 1\) intervals where the leaves represent intervals and the internal node represent the union of intervals of the leaves in its subtree. Your data structure should have size \(O(n \log n)\).

Your data structure should handle the following operations:

(a) Insert(S) in \(O(\log m)\) time where \(S\) is a new segment.

(b) Delete(S) in \(O(\log n)\) time where \(S\) is a segment.

(c) Report(r): report all segments containing \(r \in \mathbb{R}\) in \(O(K + \log n)\) were \(K\) is the number of such segments.

(d) You should also describe how to perform rotations on the tree in constant time.

3. Now that we have a data structure to maintain our live rectangles we need to also maintain for each node of the tree the length of the union of the rectangles that intersect it’s interval. There is a trick on what the exact definition that works. Let \(N\) be a node, \(P(N)\) it’s parent, and \(I(N)\) the interval of \(N\).
**Definition 0.2** \( L(N) = \text{length of the union of } S \cap I(N) \text{ where } S \text{ is a live interval and } I(P(N)) \not\subseteq S. \)

Show how to maintain \( L(N) \) under insertions and deletions in \( O(\log m) \) time.

4. Use the previous steps to give an algorithm to compute the area of \( n \) rectangles in \( O(n \log n) \).

(30) 8. *Output Sensitive Convex Hull*

In class we analyzed two convex hull algorithms - merge hull and a randomized incremental algorithm, both of which ran in \( O(n \log(n)) \). We will now describe and analyze an output sensitive convex hull algorithm. If the number of vertices determining the boundary of \( CH(P) \) is \( h \), we want an algorithm that runs in \( O(n \log(h)) \).

- Let \( p \in \mathbb{R}^2 \) and let \( P \) be a convex polygon in \( \mathbb{R}^2 \) on \( m \) vertices. Define a tangent from \( p \) to \( P \) as a directed line \( l \) between \( p \) to \( q \in P \) such that all \( x \in P \) lie to the left of \( l \). Prove that \( l \) is uniquely defined, and give an algorithm to find \( l \) in \( O(\log(m)) \) time. State any assumptions on how \( P \) is stored.

Suppose a ‘little birdie’ tells us \( h \) for a point set \( S \), \( |S| = n \). Consider the following divide and conquer convex hull algorithm. Our algorithm will divide \( S \) into \( n/h \) sets, each of size \( h \) and compute the convex hull of the smaller sets, and finally merge the results.

- Suppose \( p_l \) is the leftmost point of \( S \). Assuming the sub-hulls for each of our \( n/h \) pieces of size \( h \) have been computed, show how to find the next edge of the convex hull of \( S \) in \( O((n/h) \log h) \) time. [Hint: think of a very simple search algorithm].

- What is the overall complexity of computing the convex hull of \( S \) by repeated application of the previous find-edge procedure? Be sure to explain your result.

Unfortunately we’re basing all of our analysis on this ‘little birdie’ who so kindly told us the value of \( h \). We will now see how to remove this dependence, and still meet the same bounds.

Suppose we start with \( h' \), an initial guess of \( h \) where \( h' < h \). Upon running the divide and conquer algorithm described above with \( h' \) we will quickly see in the reconstruction step that we need more than \( h' \) edges in order to construct the global hull. Instead of running the reconstruction procedure for all \( h \) steps, let’s terminate the algorithm once we observe that the global hull requires more than \( h' \) edges.

- Suppose that initially, \( h' = 3 \), and that each time we notice that \( h' \) is incorrect, we square \( h' \) and restart the algorithm. Clearly this new algorithm terminates, as in the final iteration, \( h' < h^2 \), and we shall fully reconstruct the global hull. Prove the running time of this procedure is \( O(n \log(h)) \).