CS603: Programming Assignment

Date : April 8, 2025. Geometric Algorithms, Spring 2025

Due: 11:59:59 pm, April 20, 2025

Please read the instructions below carefully.

1. Do not copy your code from another student or from the internet. Do not use any third-party libraries.
2. No submissions will be accepted beyond the mentioned deadline.
3. Submissions should be made on Moodle.
4. There is no restriction on the programming language to be used for implementing the algorithms.
5. For both problems, the input would be given as an array. A point is represented by a pair of values while a line segment is represented by a pair of points.
6. Make sure that your code outputs the answer in sorted format.
7. Name your submission folder as "roll.zip", where roll denotes your roll number. For example, if your roll number is 203145, then name the submission folder as "203145.zip".
8. Inside the main submission folder, there should be 2 sub-folders: name them "(A)" and "(B)" respectively. Put the source code for each question in that respective folder.
9. Include a script file in each question folder which takes appropriate input and runs your code. Name that file "test.sh".
10. Make sure your code is well-documented and readable. Make a file to document your approach and prepare a code demo.

1. **(50 = 25+25 points)** Let P be a set of n points in $\mathbb{R}^2$.

   (a) Design an algorithm to compute the convex hull of P in $O(n \log h)$ time, where h is the number of points on the convex hull.

   (b) Now consider a dynamic setting where, at each step $i \in \mathbb{N}$, a point p_i is either inserted or deleted. Design a data structure to maintain the convex hull efficiently and support point containment queries (i.e., if a point p is contained in the convex hull).
   Operations: Insert a point, delete a point, and query whether a given point lies on the boundary of the current convex hull.
   Output: For each query, report `YES` if the point is on the convex hull boundary, and `NO` otherwise.
   Time complexity: Each operation should run in polylogarithmic time.

2. **(50 = 25+25 points)** Let P be a set of n points in $\mathbb{R}^d$. We say that a point $p \in \mathbb{R}^d$ *dominates* a point $q \in \mathbb{R}^d$ if for every coordinate $i \in \{1, 2, \ldots, d\}$, we have $p_i > q_i$. A point p is called *maximal point* if no other point in P dominates it. The goal is to report all the maximal (or skyline) points in P.

   (a) Design an algorithm to compute the skyline points in $O(n \log^d n)$ time. *(Hint: Use range trees.)*

   (b) Improve the algorithm for the case when the input points lie in $\mathbb{R}^3$, and compute the skyline points in $O(n \log n)$ time.

3. **(50 = 25+25 points)** Let P be a set of n points in $\mathbb{R}^2$.

   (a) Implement a range tree data structure for P to support orthogonal range reporting queries.
   Input: A static set of n points in $\mathbb{R}^2$, followed by a sequence of axis-aligned rectangular range queries.
   Output: For each query rectangle, report all points from P that lie inside it.
   Time complexity: $O(n \log n)$ preprocessing and $O(\log^2 n + k)$ query time, where k is the number of reported points.

   (b) Extend your implementation to handle dynamic updates. At each step $i \in \mathbb{N}$, a point p_i is either inserted into or deleted from the set.
   Operations: Insert a point, delete a point, and query with an axis-aligned rectangle.
   Output: For each query, report all points in the current set that lie inside the query rectangle.

4. **(50 = 25+25 points)** Let P be a set of n points in $\mathbb{R}^2$.

   (a) Implement a k-d tree data structure for P to support orthogonal range queries.
   Input: A static set of n points in $\mathbb{R}^2$, followed by a sequence of axis-aligned rectangular range queries.
   Output: For each query rectangle, report all points from P that lie inside it.
   Time Complexity: $O(n \log n)$ preprocessing and $O(\sqrt{n} + k)$ query time, where k is the number of reported points.

   (b) Extend your implementation to handle dynamic updates. At each step, a point may be inserted or deleted.
   Operations: Insert a point, delete a point, and query with an axis-aligned rectangle.
   Output: For each query, report all points in the current set that lie inside the query rectangle.

5. **(50 = 25+25 points)** Let $\mathcal{S}$ be a set of n line segments in the plane.

   (a) Compute all intersection points among the segments in $\mathcal{S}$.
   Input: A set of n line segments in the plane.
   Output: The set of all intersection points between segments in $\mathcal{S}$.
   Time Complexity: $O(n \log n + k)$, where k is the number of intersections.

   (b) Now consider a dynamic setting where, at each step, a line segment ℓ_i is inserted into or deleted from the set. Adapt the line segment intersection algorithm to support dynamic updates.

Operations: Insert a segment, delete a segment, and report all current intersection points.
Output: After each update, maintain and report the current set of intersection points.

6. **(50 = 25+25 points)** Let P be a set of n points in $\mathbb{R}^2$.

   (a) Compute a well-separated pair decomposition (WSPD) of P using a separation constant $s > 0$.
   Input: A set P of n points in $\mathbb{R}^2$, and a separation constant s.
   Output: A WSPD of P consisting of $O(n)$ well-separated pairs.
   Time complexity: $O(n \log n)$.

   (b) Use the WSPD from part (a) to compute a geometric spanner for P with stretch factor $t = (1 + \varepsilon)$.
   Input: A WSPD of P and a parameter $\varepsilon > 0$.
   Output: A graph on the point set P that is a t-spanner.
   Time complexity: $O(n \log n)$, assuming ε is constant.

7. **(50 = 25+25 points)** Let P be a set of n points in $\mathbb{R}^2$.

   (a) Implement a range tree data structure for P to support orthogonal range reporting queries.
   Input: A static set of n points in $\mathbb{R}^2$, followed by a sequence of axis-aligned rectangular range queries.
   Output: For each query rectangle, report all points from P that lie inside it.
   Time complexity: $O(n \log n)$ preprocessing and $O(\log^2 n + k)$ query time, where k is the number of reported points.

   (b) Enhance your range tree implementation by applying *fractional cascading* to improve query efficiency.
   Output: Support the same range queries as in part (a), but with improved query performance.
   Time complexity: $O(n \log n)$ preprocessing and $O(\log n + k)$ query time.

8. **(50 = 25+25 points)** Let P be a set of n points in $\mathbb{R}^2$.

   (a) Compute the maximum subset of collinear points in P using point-line duality.
   Input: A set of n points in $\mathbb{R}^2$.
   Output: The largest subset of points in P that lie on a common line.

   (b) Assume the input is in general position (i.e., no three points are collinear). Compute the triangle of minimum area that can be formed by any three points in P, using duality.
   Input: A set of n points in $\mathbb{R}^2$ in general position.
   Output: The triangle of minimum area formed by any triple of points in P.
   Time complexity: $O(n^2 \log n)$.

9. **(50 = 25+25 points)** Let P be a set of n points in the plane.

   (a) Find a unit square (axis-aligned, of side length 1) that contains the maximum number of points from P.
   Input: A set $P \subset \mathbb{R}^2$ of n points.
   Output: An axis-aligned unit square that contains the largest possible subset of points from P.
   Time complexity: $O(n \log n)$ time.

(b) Maintain the above structure dynamically: at each step, a point is inserted into or deleted from the point set.
Input: A dynamic sequence of point insertions and deletions.
Output: After each update, report a unit square that contains the maximum number of current points.

10. **(50 = 25+25 points)** Let P be a polygon with n vertices in the plane.

   (a) Let P be a simple polygon. Compute a triangulation of P.
   Input: A simple polygon P with n vertices given in counterclockwise order.
   Output: A triangulation of P into non-overlapping triangles whose union equals P.
   Time and space complexity: $O(n \log n)$ time and $O(n)$ space.

   (b) Now assume P is a polygon with holes. Implement an efficient algorithm to triangulate such a polygon.
   Input: A polygon P with h holes and a total of n vertices (including holes), each specified as a simple polygon.
   Output: A triangulation of P into non-overlapping triangles that respects the holes.
   Time complexity: Subquadratic time.

11. **(50 = 25+25 points)** Let S be a planar subdivision formed by n non-intersecting line segments.

   (a) Construct a point location data structure for S using vertical decomposition.
   Input: A planar subdivision S defined by n non-crossing segments.
   Output: A data structure that, given a query point q, reports the face of the subdivision containing q.
   Time complexity: $O(n \log n)$ preprocessing time and $O(\log^2 n)$ query time.

   (b) Improve the performance using a trapezoidal map and a randomized incremental construction.
   Input: Same as in part (a).
   Output: A more efficient point location structure using a trapezoidal decomposition of the subdivision.
   Time complexity: $O(n \log n)$ expected preprocessing time and $O(\log n)$ expected query time.
   Additional Instructions: Use a randomized incremental algorithm to build the trapezoidal map and its corresponding search structure.

12. **(50 = 25+25 points)**

   (a) Given a set of points P in the plane, implement the quadtree data structure.
   Input: A finite set P of points in $\mathbb{R}^2$.
   Output: A quadtree decomposition of the bounding square of P, with each leaf cell containing at most a constant number of points.

   (b) Let S be a set of disjoint, octilinear polygons (edges axis-aligned or at 45° angles), each with integer coordinates, lying within a square $Q = [0, U] \times [0, U]$. Given that the total perimeter is $p(S)$, compute a valid adaptive triangular mesh.
   Input: A set S of disjoint octilinear polygons with total perimeter $p(S)$, all contained in square Q.

Output: An adaptive triangular mesh that conforms to S.
Performance: $O(p(S)\log_2 U)$ time, with $O(p(S)\log U)$ triangles in the mesh.
Hint: Use a compressed quadtree-based refinement.

13. **(50 = 25+25 points)** Let P be a set of n points in $\mathbb{R}^2$.

    (a) Compute the Delaunay triangulation of P.
    Input: A set P of n points in general position (no three collinear, no four cocircular).
    Output: The Delaunay triangulation of P.
    Time complexity: $O(n \log n)$ time.

    (b) Now assume dynamic input: at each step, a point p_i is either inserted into or deleted from the current point set. Implement an efficient data structure to maintain the Delaunay triangulation under such updates.
    Input: A dynamic sequence of point insertions and deletions.
    Output: A data structure that maintains the Delaunay triangulation of the current set of points.
    Time complexity: $O(\log n)$ expected time per insertion and $O(\log^2 n)$ expected time per deletion.

14. **(50 = 25+25 points)** Let L be a set of n lines in $\mathbb{R}^2$.

    (a) Compute the arrangement $\mathcal{A}(L)$ using incremental construction.
    Input: A set L of n lines in general position (no two parallel, no three concurrent).
    Output: The full arrangement $\mathcal{A}(L)$, i.e., the subdivision of the plane induced by the lines.
    Time complexity: $O(n^2)$ time.

    (b) Given another line ℓ, report the set of cells in $\mathcal{A}(L)$ that are intersected by ℓ.
    Input: The arrangement $\mathcal{A}(L)$ and a query line ℓ.
    Output: All faces (cells) of the arrangement that intersect ℓ.
    Time complexity: Proportional to the number of intersected cells, plus logarithmic overhead if a point location data structure is used to begin the walk along ℓ.

15. **(50 = 25 + 25 points)** Let p_{start} and p_{goal} be two points in $\mathbb{R}^2$, and let $\mathcal{O}$ be a set of polygonal obstacles in the plane.

    (a) Compute the shortest collision-free path for a point robot moving from p_{start} to p_{goal}.
    Input: A set $\mathcal{O}$ of polygonal obstacles and two points $p_{\text{start}}, p_{\text{goal}} \in \mathbb{R}^2 \setminus \mathcal{O}$.
    Output: A shortest path from p_{start} to p_{goal} that avoids all obstacles.

    (b) Now assume that the robot is a unit disk instead of a point. Modify the algorithm to account for the robot's geometry and compute a collision-free path.
    Input: Same as in part (a), but the robot is modeled as a unit disk.
    Output: A shortest path from p_{start} to p_{goal}, avoiding collisions while accounting for the robot's shape.

16. **(50 = 25 + 25 points)** Let S be a set of n point robot locations in $\mathbb{R}^2$.

(a) Find a point $q \in \mathbb{R}^2$ that minimizes the sum of Euclidean distances from q to all points in S.
Input: A set $S = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$.
Output: A point q minimizing $\sum_{i=1}^{n} \|q - p_i\|$.

(b) Compute the minimum enclosing disk of S, i.e., the smallest disk that contains all the points.
Input: The same set S of n points.
Output: A disk of minimum radius that encloses all points in S.
Time complexity: $O(n)$ expected time using a randomized incremental algorithm (e.g., Welzl's algorithm).