**CSL 852, Computational Geometry: Practice Problems**

# The plane sweep technique and applications

1. Design a polyhedron (3 dimensional version of a polygon) such that guards placed at every vertex may not be able to cover the entire interior.

2. Design an $O(n \log n)$ algorithm for the three dimensional maximal problem for an input set $S = (x_i, y_i, z_i)$ $i \leq n$.
   Hint: Use a generalization of line sweep and maintain the information about the 2D maximal layer.

   Solution: Sort and look at all points in decreasing order of z-coordinate. For the points previously considered, maintain the 2-dimensional maximal layer in a binary search tree. The search tree is built on the x-coordinates of the points on 2D Maximal Layer. When a point $p_i$ is considered, find the point of tree, say $p_k$ with $x$ just greater than $x_i$.
   If $y_i > y_k$, insert $p_i$ in the tree and report it as a maximal point. Now repeatedly find the point, say $p_j$ with $x$ just less than $x_i$ in the tree. If $y_i > y_j$, remove point $p_j$ from tree and repeat until such points can be found.
   Since each point is inserted and deleted atmost once in the tree, so total time can be bounded by $O(n \log n)$.

3. Design and **implement** an $O(n \log n)$ algorithm for computing the area of the union of $n$ isothetic (axis parallel) rectangles.
   Note: The choice of the language is yours.

   Solution: Line Sweep can be used to find the area of union of n isothetic rectangles.
   Sort all vertical line segments of all rectangles according to the $x$-coordinate.
   Move in the order of increasing $x$ and as we encounter a line segment, make changes in the interval of $y$ that the union of rectangles cover until the next line segment. This $y$ interval is updated by inserting or deleting the interval of line segment from an interval tree data structure. This step takes $O(\log n)$ at each step. Also at each event point, we do $Area = Area + (y \triangle x)$ where $y$ is the $y$-interval covered between the last and this event point and $\triangle x$ is the difference in $x$ of both points. We get total area after all event points are exhausted. Since total event points are $n$, total running time of algorithm is $O(n \log n)$.

4. Given a set $S$ of non-intersecting line segments, design a data structure that supports the following query: For any axis parallel rectangle $R$, it returns the set of line segments $S' \subset S$ that intersects the rectangle $R$ (including the interior). The query should be answered in $O(polylog + k)$ time where $k$ is the number of segments reported and the space should be near linear.

5. Given a set $S$ of $n$ line segments (mutually non-intersecting), construct a data structure that supports a query of the following kind -
   For any arbitrary point $p$, find the segment that lies immediately above $p$ (called vertical ray shooting query)
   (i) Design a data structure based on segment trees that answers such queries quickly.
   (ii) **Bonus** Can you present a scheme that answers the queries in $O(\log n)$ time ?
   Solution: Construct a segment tree and within each node $v$ build a data structure that supports vertical ray shooting query for the segments stored in $v$, sy $S_v$. Since the segments in $S_v$ are totally ordered (within the interval spanned by $v$), we can do binary search using a *above-below* primitive in $O(\log n)$ steps. So vertical ray shooting is done by first using a binary search in the $x$ direction that

identifies all those nodes $V$ such that $\cup_{v \in V} S_v$ are exactly those segments that intersect the vertical line through the query point $q$. The set $V$ is simply the search path of $q$ in the segment tree. Then we do binary searches in all nodes of $V$ and report the closest segment in the upward direction. Overall it takes $O(\log^2 n)$ for query and $O(n \log n)$ space.

To improve the bounds, you can use line sweep to build the trapezoidal map of the set of line segments using vertical visibility information during the line sweep process. Then build the Dobkin-Kirkpartick planar point location data structure that can answer a query in $O(\log n)$ time. If we know the trapezoid, we also know the vertical visibility segment. The preprocessing takes $O(n \log n)$ time and the space is $O(n)$ - the total size of all trapezoids.

6. Given a set of $n$ numbers, design a $O(n \log n)$ algorithm for computing the number of *inversions*. Two numbers $x_i, x_j$ are *inverted* if $i < j$ and $x_i > x_j$.

   Solution: Merge Sort with a following modification gives the number of inversions in a list of n numbers. Divide the given list into 2 halves. Sort each half recursively and merge them to give final sorted list along with the number of inversions.
   Base Case (Length of list is 2) : If $x_1 < x_2$, then return list and number of inversions = 0, else return $(x_2, x_1)$ and number of inversions = 1.
   Inductive Case : Let the left sorted list be $(y_1, y_2, ..., y_{n/2})$ and their number of inversions = $n_L$ and right sorted list be $(z_1, z_2, ..., z_{n/2})$ and their number of inversions = $n_R$. Initialize an integer $t = 0$ and start by placing pointers at $y_1$ in list 1 and $z_1$ in list 2. If $y_1 < z_1$, copy $y_1$ to the new merged list and take pointer to $y_2$. Else copy $z_1$ to merged list and update $t = t + n/2$. If currently, $y_i$ and $z_j$ are compared and if $z_j < y_i$, then $z_j$ becomes next element of merged list and $t = t + (n/2 + 1 - i)$. When either of the list $y$ or $z$ gets exhausted, rest of elements in other list are copied onto the merged list and total number of inversions = $n_L + n_R + t$. Since the merge step requires $O(n)$ comparisons and arithmetic operations, so the recurrence for running time is same as merge sort and hence time is $O(n \log n)$.