Practical Computational Geometry Algorithm Exercises

EXERCISE 1

PROBLEM 1 (TOP-1 SEARCH)

First find the convex hull of P, this take O(n log n).

We will then transform the points into polar coordinates $(\theta,\rho)\in[0,2\pi)\times[0,\infty)$.

We will insert the point based on order of θ into a binary search tree with it's associate ρ.

Given a linear preference function we can simple transform into polar coordinates.

This gives the general direction θ of where top-1 is, then simply do searching on the binary tree to find the nearest neighbor towards direction θ.

PROBLEM 2(MERGING CONVEX HULLS)

CH(A) and CH(B) is store clockwise.

Orient(p,q,r) > 0 → Counterclockwise orientation of p, q, r.

Orient(p,q,r) < 0 → Clockwise orientation of p, q, r.

Orient(p,q,r) = 0 → Colinear of p, q, r.

Algorithm:

```
Let a be the rightmost point of CH(A).
Let b be the leftmost point of CH(B).
While (orient(b,a,a.succ) >= 0 AND orient(a,b,b.pred) <=0)
    While (orient(b,a,a.succ) >= 0) , a ← a.succ (move a clockwise)
    While (orient(a,b,b.pred) <=0), b ← b.pred (move b counterclockwise)
Return ab.
```

PROBLEM 3(MERGING CONVEX HULLS (AGAIN))

```
Let $p_1$ be bottommost point of P=P1+P2 and $p_0$ be $(-\infty,0)$.
For k from 1 to (max(|P1|,|P2|) = h)
    For i from 1 to 2
        For each vertex $q_i$ inside CH($P_i$), compute the maximum angle $p_{k-1}p_kq_i$.
```

Then we have q_1 and q_2, that maximizes angle $p_{k-1}p_kq_i$ for $CH(P_i)$ respectively.

Let $p_{k+1} \leftarrow q$, such that q is either q_1 or q_2 that maximizes angle $p_{k-1}p_kq$.

If $p_{k+1} = p_1$, return $\{p_1, \dots ,p_k\}$ our final convex hull

EXERCISE 2

PROBLEM 1 (GENERAL BINARY SEARCH) (DONE)

Denote max(A) be the maximum values of the dataset A.

Denote L be the leftmost data in the array, M be the middle data in the array, R be rightmost.

Algorithm:

We will be cut the dataset A in the middle like binary search, we will find the maximum value of L, M and R. This only take 2 comparisons.

There will be 3 cases occur.

Case 1: If L is the maximum value out of the 3 values.

In this case, the max(A) will be in the first half of A.

Then, just repeat the algorithm on the first half of A and ignore second half of A.

Case 2: If R is the maximum value out of the 3 values.

In this case, the max(A) will be in the second half of A.

Then, just repeat the algorithm on the second half of A and ignore first half of A.

We will proof that the maximum values of dataset for case 1 and 2 must be on the respective half.

Let's assume max(A) is in the second half of A instead of the first half for case 1.

As case 1 says L is the maximum between L, M and R. We must have $L > R$ and $L > M$.

Based on our assumption and the pattern of the array, all successors of max(A) is decreasing until a turning point and the array is increasing again. That is to say max(A) > R.

To maintain the pattern of the data, R must be a value between max(A) to L. However, this violates that L is the maximum values out of L, M and R. By contradiction, max(A) must be on first half of A.

This is symmetric for case 2, I will skip the proof.

Case 3: M is the maximum value out of the 3 values.

If successor of M > M, we can conclude that max(A) is in second half of A and repeat the algorithm on the second half of A. It is not possible for max(A) to be in the first half, as it will violate the (shifted) bell shape of the array.

Otherwise (predecessor of M > M), repeat the algorithm on the first half of A.

Repeatedly run the algorithm until max is found when >= 3 elements left in the array (brute force compare 3 values).

PROBLEM 2 (GIFT WRAP)

As the line l turn anticlockwise, the first vertex hit must be on the upper hull and will be on the top most vertex or one of the predecessor of the top most vertex.

For each vertex q (on the first quadrant), Compute angle lpq and store in an array.

The first vertex hit will be the vertex that make the smallest angle.

PROBLEM 3 (GIFT WRAP AGAIN)

Essentially apply algorithm on problem 2 on each of the convex polygon.

Then we will have a set of vertices that make the smallest angle w.r.t it's convex polygon and the vertex that make the smallest angle lpq wins.

PROBLEM 4 (OUTPUT-SENSITIVE CONVEX HULL)

We will be splitting the whole dataset into n/k subsets. Then, compute the mini-convex hull of each of these small subsets. This will take O(k log k) time for each small subset and O(n log k) time to compute every subset's convex hull and store in an ordered array.

After that, we will need to merge all those convex hulls back together to form one big convex hull of the dataset(basically Jarvis algorithm). Starting from p1 be the most bottom point of the whole dataset and p0 be (-infinity, 0).

For h to 1 to k;

For i to r, compute angle $p_{h-1}p_hq_i$. The q_i that gives the largest angle is what we want.

Let p_{k+1} be the q_i that maximizes angle $p_{h-1}p_hq_i$.

If $p_{k+1} = p_1$, the algorithm is finished here and return <p1,…,p_k> as our final convex hull.

Total number of Jarvis runs is at most k, so the total running time for the for loop stage will be O(n log k). Given the first stage also take O(n log k), the total running time would be O(n log k).

EXERCISE 3

PROBLEM 1 (RANGE MAX)

k-d tree

We do splitting with max spread, that is we find the maximum spread in the 2 dimensions x and y.

Then, we will be cutting at the middle of the maximum spread, that is cut at (max_spread /2) on the respected dimension.

Algorithm:

```
Algorithm BuildKdTree(P,depth)
1. if P contains only one point
        2. then return a leaf storing this point
        3. else if depth is even
                4. then Split P with a vertical line ` through the median x-coordinate into P1 (left of or on
        `) and P2 (right of `)
                5. else Split P with a horizontal line ` through the median y-coordinate into P1 (below or
                on `) and P2 (above `)
                 6. vleft ← BuildKdTree(P1,depth+1)
                 7. vright ← BuildKdTree(P2,depth+1)
                8. Create a node v storing `, make vleft the left child of v, and make vright the right child
                of v.
                9. return v
```

Procedure name SEARCHKDTREE(v,R)
Input: The root of (a subtree of) a kd-tree, and a range R.
Output: All points at leaves below v that lies in range.

```
1: if v is a leaf
2: then Report the stored at v if it lies in R
3: else if region(lv(c)) is fully contained in R
4:                then REPORTSUBTREE(lc(v))
5:                else if region(lc(v)) intersects R
6:                then SEARCHKDTREE(lc(v),R)
7:        if region(rv(c)) is fully contained in R
8:                then REPORTSUBTREE(rc(v))
9:                else if region(lc(v)) intersects R
10:               then SEARCHKDTREE(lc(v),R)
```

Simply search for the point within the query range and return the point with maximum y coordinate in the range query result.

PROBLEM 2(BATCHED LINE DRAGGING)

We will apply k-d tree to points in P, then we do rectangle range check for each segment. The segment will be the rightmost line of the rectangle, the leftmost bound of the rectangle can be the leftmost point for P.

After the rectangle query is done , simply find the point with the largest x coordinates values, if the result is empty, return nil.

PROBLEM 3(ROTATING SWEEP)

Let’s assume p is at [0,0]. (Everything can be shift to an arbitrary place)

Given all endpoints of segments, we can find their polar coordinate $(\theta,\rho)\in[0,2\pi)\times[0,\infty)$, sort by degree.

We shoot a ray at p towards direction θ, denoted R(θ). We will rotate this ray anticlockwise.

We will then calculate the intersection point of the ray and segments and return segment index by their order on R(θ) (compare ρ, BST for insertion).

Whenever a unique segment is place at the first place of the array (left most leaf of BST), we return it.

Events:

Hit start of segment: Calculate the intersection point of the ray and segments and insert segment index to the BST.

Middle of segment: Usually no update is needed as we only store the relative distance to p from segment.

Hit end of segment: We will delete the entry inside the BST.

EXERCISE 4

PROBLEM 1 (POLYGON INTERSECTION) (DONE)

We first separate P_1 and P_2 into upper hull and lower hull.

Let’s focus on the upper hulls, as it is symmetric for lower hulls case.

We will sweep from left to right.

Algorithm:

We will compare the x coordinates of p_1 and p_2 on the upper hull of P_1 and P_2.

Advance the vertex (update p_1 or p_2) with smaller x coordinate value by using successor.

If the advanced vertex passes the vertex on the other upper hull, check if there is intersection or not.

If there is intersection and it wasn’t reported before, report it. Otherwise, move on.

PROBLEM 2 (POLYGON INTERSECTION, AGAIN)

Like problem 1.

We first separate P_1 and P_2 into upper hull and lower hull.

Let's focus on the upper hulls, as it is symmetric for lower hull case.

We will sweep from left to right.

Algorithm:

First, we will indicate the convex hull with larger x coordinates of leftmost point as inner_hull.

We will compare the x coordinates of p_1 and p_2 on the upper hull of P_1 and P_2.

Advance the vertex (update p_1 or p_2) with smaller x coordinate value by using successor and we will report the vertex if advance vertex is on the inner_hull.

If the advanced vertex passes the vertex on the other upper hull, check if there is intersection or not.

If there is intersection and it wasn't reported before, report it and switch the inner_hull to the other convex hull. Otherwise, move on.

PROBLEM 3 (POINT IN POLYGON) (DONE)

First introduce a method Line(point 1, point 2, point 3) to see which side point 1 is on given a line passes point 2 and 3. This is easily be done as we learn about halfplane. Let say it output 1 for point 1 on upper halfplane and 0 for lower halfplane.

Let's denote CP[i] be the i-th element in the convex polygon array

Algorithm:

```
If convex polygon has 3 vertices, it is easy to see if a point is inside the triangle or not.

If (Line (query point, CP[0], CP[n/2])>0)

        Repeat the algorithm with the query point and a new Convex Polygon with vertices CP[n/2] to
        CP[n] and CP[0].

Otherwise,

        Repeat the algorithm with the query point and a new Convex Polygon with vertices CP[0] to
        CP[n/2].
```

The idea is like cutting the convex polygon in half (or number of vertices), and see if the point fall on the upper or lower halfplane with respect to the cutting line. Until number of vertices are small enough to brute force it.

PROBLEM 4 (CONVEXITY DETECTION)(DONE)

Idea: In clockwise order, each 3 points subsets of the whole set should be turning clockwise for a polygon to be convex. If a anticlockwise turn appear, it must not be convex violate definition of convex hull.

Algorithm:

Start with arbitrary 3 consecutive point and use orient(p,q,r) method inside lecture notes. Storing the first vertex.

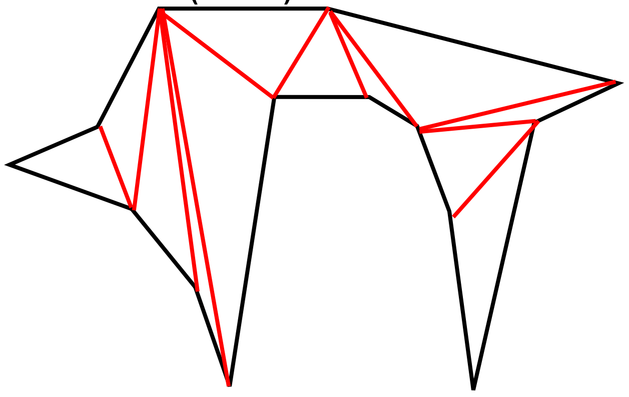
If orient(p,q,r) < 0, that means these 3 points in clockwise order.

> We will then move to the next 3 consecutive point by deleting p and get the successor of r and repeat the process.

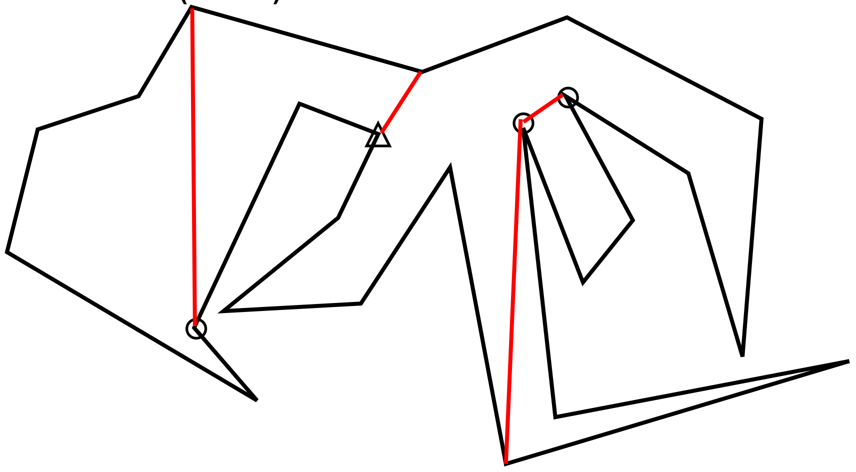
If we reach back the first vertex without having orient(p,q,r) > 0 happening, conclude it is convex. Otherwise conclude not convex.

EXERCISE 5

PROBLEM 1 (DONE)



PROBLEM 2(DONE)



PROBLEM 3

Imagine we shoot a ray from the point to the right infinity, this is to ensure every edge will be encounter by this ray.

If the number of intersections found is odd, we conclude that the point is inside the polygon.

If the number of intersections found is 0 or even, we conclude that the point is outside the polygon.

EXERCISE 6

PROBLEM 1(DONE)

Half-planes **e and d** will recurses into a 1D instance.

PROBLEM 2(STAR-SHAPE DETECTION)(DONE)

We will view every edge as a halfplane and we have a linear programming with n edges.

A polygon is star shaped if and only if the result of the linear program is non-empty (Kernel of the polygon is not empty). That this 2D linear programming takes O(n) to find feasible region by randomized incremental algorithm.

PROBLEM 3

EXERCISE 7

PROBLEM 1

Part 1:

At first, we will have circle pass a and b.

Process c, we will move to 2-point-fixed-MEC, and return circle passes a b and c.

Process d, nothing need to change.

Process e, it is not in C. So move to 1-point-fixed-MEC and C passes e and a doesn't enclose b. so move to 2-point-fixed-MEC({a,b,c,d,e},{e,b}) , it return circle passes b and e. Then everything is contained inside this circle.

Part 2:

PROBLEM 2

Part 1:

Part 2:

Simply find the max, min in both x and y directions.

Computer max-min in both directions, the larger one is the size of the square and the square will be placed on those 2 points.

PROBLEM 3

EXERCISE 8

PROBLEM 1(DONE)

Dual lines: A* : y = x – 1

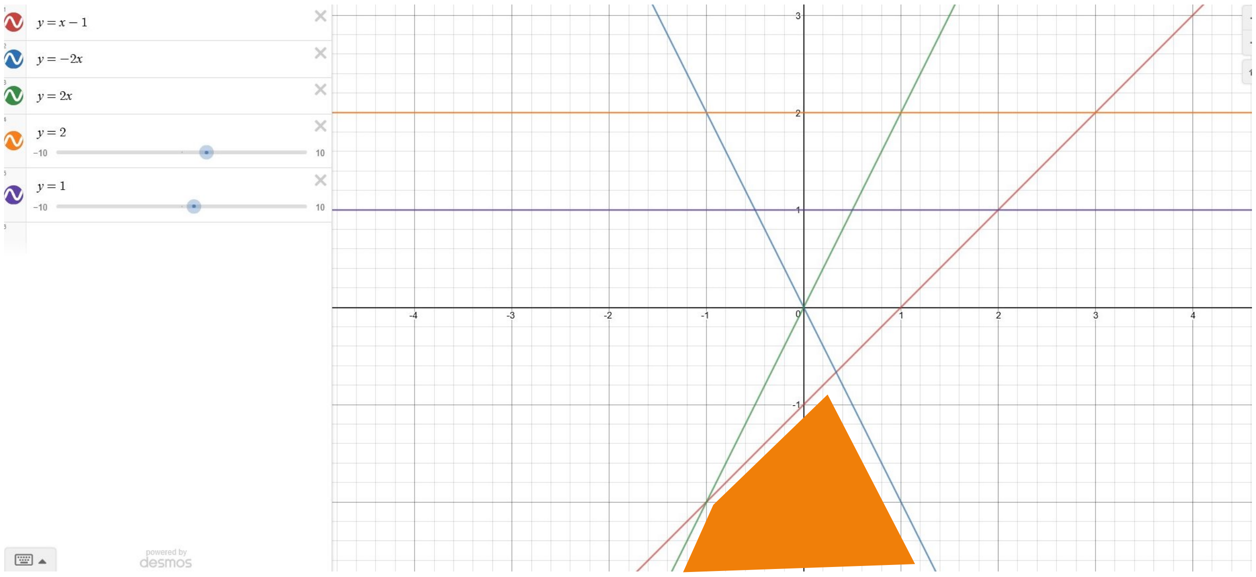
B* : y = -2x

C* : y = 2x

D* : y = 2

E* : y = 1

The orange area is the lower envelop by the dual lines C* A* B* (in that order) . Because CH of all point have the upper hull C A B (in this order)



Part 2

Consider remove A* from halfplanes, we will then solve the corresponding LP on the rest of the halfplanes.

The result p of the corresponding LP on the rest of the halfplanes is NOT below A*.

That is to say A* does contribute to the lower envelope in the dual space, and A is an extreme point of P.

Part 3

The point the instance of linear programming that ignoring A* would be (0,0) and the corresponding line in primal space is y=0. It basically ignored point A and construct the Convex Hull by connecting B and C

PROBLEM 2

PROBLEM 3

MIDTERM

PROBLEM 4 (X-MONOTONE POLYGON P AREA)

First, we set area = 0.

We will run the triangulation on the x-monotone polygon.

Every time we construct a diagonal (forming a triangle), we will computer the area of the triangle by cross product/2 and add to area.

At the end, the x-monotone polygon will be triangulate, and all triangle area will be summed up.