Homework #1

Lucio Franco, worked with Sam Windham.

10/17/2017

1)

```
PROCEDURE LeftTangent(Polygon p, Point q):
    // first point is left tangent
    if isLeft(q, p[n-1], p[0]) AND !isLeft(q, p[0], p[1]):
        return (q, p[0])

    let i = 0
    let offset = 1
    loop:
        // direction of current edge
        let edgeDir = isLeft(q, p[i], p[i+1])

        // current edge is the left tangent point
        if isLeft(q, p[i-1], p[i]) AND !edgeDir:
            return (q, p[i])

        // check directions and continue search
        let prevEdgeDir = isLeft(q, p[i - offset], p[i - offset + 1])

        // current edge is left
        if edgeDir:
            // previous edge is left AND current point left of previous point
            if prevEdgeDir AND !isLeft(q, p[i-offset], p[i]):
                i -= offset
                offset = 1
            // prev edge is right OR current edge left of previous edge
            else:
                offset *= 2
        else:
            // previous edge is left
            if prevEdgeDir:
                i -= offset
                offset = 1
```

```
        else:
            offset *= 2

    return i = min(i + offset, n - 1)
```

The algorithim will run an exponetial search to find the tagent. It will return the line segment that forms the left tagent on P. The `isLeft(a, b, c)` will check if the three vectors create a left turn. Exponential search runs in $O(logn)$.

3)

We know that there are $\binom{n}{2}$ lines inducded from a set of n points in general position. If we transform every point to the dual, the lines produced will intersect forming the point that represents the line between those two points that are now lines in the dual. We know that if the point formed from the intersection of the two lines is on the positive side of the dual, aka it has a x value greater than zero, it's line segment in the primal has a positive slope.

From this we can run a plane sweep starting at the origin and sweeping along the x axis of the dual. Each time we encounter an intersection of two lines we can count this as a positive slope. There are also four possible combinations of the lines, when at the origin one can tell if both lines have positive slope, the line above has positive slope and the line below has negative slope, the line above has negtative slope and the line below has positive slope, or they both have negative slopes.

From this we can eliminate possible events because for a fact we know that those two lines can not intersect as we move to the right when the line above has a positive slope and the line below has a negative slope. For part (a) we would continue the line sweep until weve gone through all of the intersections but for part (b) instead of starting at the origin, we would start at m' and continue until we have hit m''. Once an event has been checked, the lines can be swapped and checked against their neighbors.

Since as we move to the right the slope of the line produced from the intersection in the dual will have a larger slope. Every intersection found within this bound will be counted as a positive slope. A brute force algorithinm would take $\theta\binom{n}{2}$ but with the plane sweep and the eleimination of events we can reduce this. We know that for n points, there can be k induced lines with positive slope, with a max of $\binom{n}{2}$. Each event we face takes $O(logn)$ using a balanced binary tree. To initally build the status it takes $O(nlogn)$. Therefore the runtime will be $O(nlogn) + O(klogn) = O((n+k)logn)$.

4)

Let $R(\alpha)$ be the ray from p with the angle α. Now we can create a schedule with each endpoint sorted on α. Now a min-heap is created to store the visible elements, first we add the first event to the min-heap. Now on the other events either a segment is removed or is added. The heap is sorted on the minimum distance from p. Since we know that the segments are disjoint, the order in the heap will not change.

Therefore when process and event, there are two possiblities, the segment that belongs to the event is in the heap or its not. If it is in the heap we remove it, if it is not in the heap we can add that segment. At the end of each heap operation, the heap is peaked in $O(1)$ and that segment is reported as visible.

There are a possible $2n$ events because each segment has two endpoints. At each event there is either a heap insertion or a deletion which can be performed in $O(logn)$, the minimum of the heap is also peaked in $O(1)$. Therefore the runtime is $O(nlogn) + O(nlogn) + O(nlogn) = O(nlogn)$. This is because we need to initially sort the endpoints, we need to process each in the schdule and to sort and remove duplicate visible segments, therefore the runtime would be $O(nlogn)$.

5)

To solve this problem a randomized incremental approach can be used. Initally we start by selecting two points at random. At this iteration we can form the line that goes through orthogonally the midpoint of the line segment formed from the two points. Now we can build a hashtable that will store all points with in r where r is the nearest distance to the line from a point. We can bucket them via x/r and y/r similar to the 2D bucketing solution from homework 1.

Now that we have built the hashtable, we can now randomly pick another point and place it in the hashmap, if there is an item within the hashtable at the same location then we know there can not be a new line l formed because the points within the bucket are too close to redo the soution. If there isnt a point within the same bucket we check all eight neighbors, if we find a point within those buckets, we then run a L2 test to see if the new added point is further away from the point in the bucket we are checking than the nearest point currently from l. If it is then we must form a new line. If there is a point within L2 distance the new solution can not be better than the previous therefore we can keep the previous.

To form the new line we find the midpoint between the new two farthest points, and place a temporary line. We can now check along this line that there are no other points that are closer to it than the two points we chose to create the line. If there is a point that makes a left turn, we can rotate the line, change its angle to no longer be parallel to the two points. If this does not make the other point

that is closer than the points we have selected for the line then we revert to the old solution. If we can form this new line then we must rebuild the hashtable with the new distance as the r.

Since to build the hashmap it takes $O(i)$ and we must loop through everypoint atleast once $O(n)$, and inserting into a hashtable is $O(1)$. In a worst case the algorithim would run in $O(n^2)$ since we may have to rebuild the hashtable everytime.