A brief solution plan for HW2

2.1

Since the $n$ segments are disjoint, we can sort them in a BST by the $x$-coordinates of either their upper endpoints, or their lower endpoints. This takes $O(n \log n)$ time.

To determine between which two segments the query point lies at, we can simply perform a binary search (BS) recursively:

Starting from the root of the BST;
If lying on the left-side of the segment, perform BS on the left sub-tree
Else, perform BS on the right sub-tree.

The query takes $O(n)$ time.

So all together, the algorithm takes $O(n \log n)$ time.

2.2

The plane sweep plan is almost the same as described in book. The only difference is that once we detect an intersection, report “True” and return. Otherwise, traverse all start & end events in the event queue.

To initialize the event queue, we need $O(n \log n)$ time.

Each time we reach an event, the update to the status tree requires $O(\log n)$ time.

In the worst case, we detect the only intersection at the end of the sweep. The total time would be:

$O(n \log n)$ {initialization} + $2n \times O(\log n)$ {traverse all $2n$ endpoints} + $O(\log n)$ {the only intersection at the end}

Which is still in $O(n \log n)$ time.

2.5

True;
True;
False---Explanation follows:
In this subdivision, $\text{Twin} \left( \text{Prev} \left( \text{Twin} \left( \vec{e} \right) \right) \right) = \vec{e}_4$, while $\text{Next} \left( \vec{e} \right) = \vec{e}_3$.

True.

2.6

A simplest example would be an un-bounded subdivision of the plane which only excludes a line segment inside it, as shown by:

2.9

We are given the double-connected edge list, with face list and vertex list. To output all faces with vertices on their outer boundary, we have to detect the face that is not out-bounded.

```
LOOP VISIT face f in the face list
  IF f.OutComponent == NULL
    CONTINUE
  ELSE
    Edge=Edge0=f.OutComponent
    WHILE Edge.Next ! = Edge0
      Add Edge.Origin to the vertex set corresponding to face f
    Edge=Edge.Next
```
2.10

Almost like the plane sweep algorithm in textbook. The only difference is that we don’t need intersection event. The event queue is determined at the very beginning with n segments and m test points.

To sort the 2n end-points of the segments and m test point in the event queue, we need $O((n+m)\log(n+m))$ time.

The status queue is a BST, which only contains the intersected segments with the sweep line. To check in which area the test point lies in, we can use the method in Problem 1, which takes $O(\log n)$ time for each point, after determine the area, we can use the IncidentFace() method to decide the face, which cost O(1) time. So for sweeping we totally need $O((n+m)\log n)$ time.

Therefore, the whole algorithm requires $O((n+m)\log(n+m))$ time.

2.11

We can break each circle into 2 semicircles with a vertical line crossing their centers. Instead of line segments, we can perform the sweep line algorithm on the 2n semicircles. The only difference is that we have to distinguish the “intersection” of two semicircles from a common circle, and that the intersection between circles can be twice.

2.14

This is still similar to plane sweep algorithm. But instead of line sweep vertically or horizontally, we now set p as the origin, sort the polar angles of the endpoints to p, and sweep counterclockwise.

The status queue is built according to how far the intersect segments are to the origin.

Each time after update of an event, if a segment in the status queue becomes the “top” one, we will mark it as “visible”. The segments, if never “emerging” at the top of status queue from entering the queue until leaving, will not be counted. In this way, we detect all the visible segments from p.