# MAC0331 - Lista 1

Matheus T. de Laurentys, 9793714

April 25, 2020

**Q 1:**
Algorithm details: if a point is intersecting a line segment, I will say that the point is in the region on the right of the segment.

Both the segment insertion and the point look-up are based on a routine (1) that checks if a point is strictly to the left of the segment (the same used in class). Note that (1) is a *constant time* operation.

The tree is going to be a self-balancing binary search tree, and the insertion algorithm uses **any** point of the segment being inserted and (1) to decide if insertion is on the left/right of the segment it is compared to. Since it is a self-balancing binary tree, n consecutive insertions take time in $O(nlgn)$.

In the same way, a single point can be checked against the segments on the binary tree. Since it is a balanced binary tree, the height is in $O(lgn)$, and, therefore, determining the region a point blongs to is also in $O(lgn)$ since (1) is a *constant time* operation.

**Q 4:**
This is exactly what I did in PE01. I will solve it with the sweep line method as described in class. Following is python code that solves the problem. Segment takes the polygon and returns a list of its segments. verify_intersection returns whether the two segments intersect.

```
def treat_left(s, bst):
        bst.insert(s)
        ns = bst.get_neighbours(s)
        ret = verify_intersection(s, ns[0])
        if (ret) : return True
        ret = verify_intersection(s, ns[1])
        if (ret) : return True

def treat_right (s, bst):
        ret = False
        bst.remove(s)
        ns = bst.get_neighbours(s)
        if(ns[0] and ns[1]):
                ret = verify_intersection(ns[0], ns[1])
        return ret
def Scanline (segments):
        segments = segment(P1) + segment(P2)
        segments = sorted(segments, key=functools.cmp_to_key(compare_segments))
        heap, hmap = make_event_points(segments)
        bst = balanced_binary_tree(Node_Seg)
        while (not heap.empty()):
                pt = heap.get()
                for seg in pt.left:
                        if(treat_left(segments[seg], bst)):
                                return True
```

```
26            for seg in pt.right:
27                if(treat_right(segments[seg], bst)):
28                    return True
29        return False
```

**Q5:**

This is another application of the same sweep line algorithm above. I will make a change on the verify_intersection(), segment() methods. The only line cheanged is line 17.

Now, verify_intersection() will check $\Delta x^2 + \Delta y^2 < \Delta r^2$ to return whether there is an intesection. segment() will, instead of getting all the edges of the polygon, generate a single segment given by $[(x - r, y), (x + r, y)]$. Therefore, line 17 will be:

**17.** segments = [segment(D) for D in disks]

This algorithm is in $O(nlgn)$ because sorting the disks is in that class, and there is a linear amount (2n) of event points generated (each event is in $O(lgn)$).