Jason Caplan
Design and Analysis of Algorithms
Professor Leff
HW10

1. **Brute-Force Solution**
   We can check each pair of coordinates (x_1, y_1) and (x_2, y_2), calculate their local maximum area by multiplying the distance in x, $|x_2 - x_1|$, by the height of the smaller y, $\min(y_1, y_2)$. If the current pair has a greater local maximum area than the best found thusfar, we update variables to hold their indices and update max area. Checking each pair is $O(n^2)$.
2. **Core Greedy Heuristic**
   In a sentence, we first try to maximize distance in x, and then work our way "inward," shrinking x slightly, but seeking greater y by rejecting smaller heights. More specifically, we have left and right pointers which start at the left and right extremities respectively, and we increment the left pointer if the left's y value is smaller, otherwise we decrement the right pointer. Each increment or decrement we recalculate a local maximum area, and compare to the best thusfar, updating our Answer if it's better. We stop when the pointers meet.
3. **Pseudo-code**

```
findMaxRectangle(coordinates ):
    best_thusfar = Answer(maxArea=0, left-x, right-x, height)

    left_ptr = left-most-x, right_ptr = right-most-x

    while (left_ptr != right_ptr):
        area = min(left_ptr's y, right_ptr's y) * (right_ptr's x - left_ptr's x);
        if (area > maxArea of best_thusfar):
            update best_thusfar to hold area, the ptrs' x's, and min of the y's
        if (left_ptr's y was smaller):
            increment left_ptr to next x to the right;
        else:
            decrement right_ptr to next x to the left;
    return best_thusfar
```

4. **Proof**
   The key insight is that when we start with a maximized x distance, the only way for any other rectangle to have a greater area is if it increases the height (since each subsequent x distance will be smaller or equal to the original x-maxed rectangle) . And the only way to increase the height of a rectangle is to increase the smaller of the two y values. Therefore, when we start with the maximized

x, the only way any other rectangle can have a larger area is if the smaller y changes. This is true on each subsequent iteration as well – the only way to possibly achieve a greater area than the previous iteration (since each iteration shrinks the x distance) will be for the smaller y to change, so we move the corresponding pointer to avail ourselves to that possibility. To sum up, since we start with a given area, and continuously take steps that represent the only possible way to find a larger area than the previous step, we will find the area that is maximized as much as possible, thus solving the problem.
Performance-wise, it's easy to see that this is $O(n)$, because each iteration reduces the distance in x between the pointers by 1, and this distance goes from $n - 1$ at the beginning to 0 at termination (when the pointers meet).