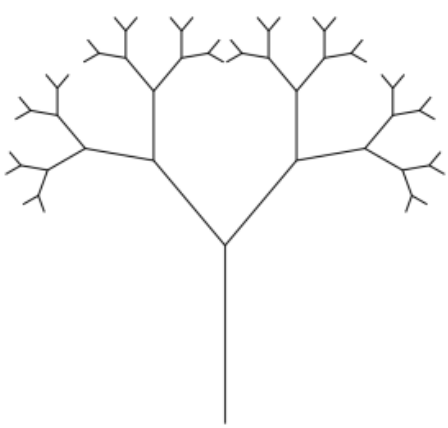
Homework 4

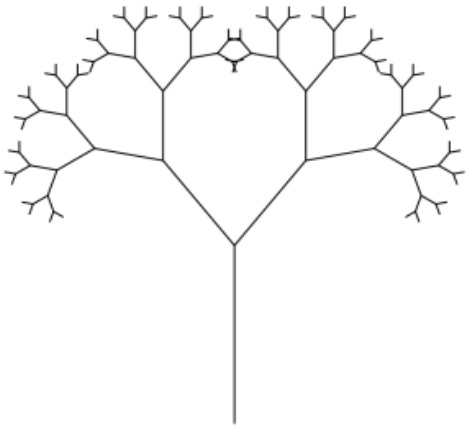
1. A = [[3], [5, 8], [9, 2, 6], [4, 7, 3, 1]], Path output: [Left, Left, Right].
   Not sure what an optimization means in this context, if it means that a computer scientist can use this algorithm instead of manual path computation, in this case it optimizes developer's lifetime and makes his life less depressive. This algorithm computes a most expensive path from a top to a bottom. Also, this algorithm doesn't use a recursion. And uses additional tables to keep values which help to determine a most expensive path.

2. For the top element we don't need to compute a path complexity because the top represents the only a single element. Also, if we pick up a specific start point, then obviously we don't have to compute complexities for his right and left neighbours. However, we don't need to compute complexities for an upper level, because we can go only right or only left. If the point for some reason is located at a corner then we need to compute complexity for only a single point, because you simply can't go to any more direction, only left or only right.
   We simply can compare (i + 1, j) with (i + 1, j + 1), if the first one is bigger then path is (i, j), (i + 1, j) otherwise (i, j), (i + 1, j + 1).

3. If we replace loops by recursion calls inside the triangle-path-dynamic function then it won't make any performance issues except additional memory consumption. If we make triangle-path recursive itself then a time complexity will be higher. **θ(2^n) > θ(n*m)**.
   Any recursive solution consumes additional memory, because it generates a same method calls with different parameters, simple loop does not do that.

   A recursive call would look like a fractal tree but upside down.





4. Substitution method
   $T(n) = 2T(n - 1) + 1 = 2(2T(n - 2) + 1) + 1 =$
   $2^2T(n - 2) + 2 + 1 = 2^2(2T(n - 3) + 1) + 2 + 1 =$
   $2^3T(n - 3) + 2^2 + 2 + 1$
   We see that with each substitution an exponentiation gets bigger by 1;
   Since we've got to find the theta notation, we simply can skip constants, then $\theta = (2^n)$.

   θ(2^n) > θ(n*m), recursive method slower than dynamic.

5. In general a principal of this problem is the same as for bottom-up, but algorithm would go from the top. I would start this algorithm from a second level because the first one is a constant. Also, I would start iteration from left to right. Additional tables I would use for storing sums.
   In my opinion triangle-path-finish is not required here, we simply can find the max value by comparing with neighbors nodes on the same level. The max node would be a transitional node.