PROBLEM 2 - DYNAMIC QUAD-SPLIT METER

(LINK)

This interactive p5.js program dynamically divides the canvas into four rectangles, with each rectangle's sides meeting at the current mouse position. Each rectangle changes its grayscale shade based on its area, and the real-time area value (in square centimeters) is displayed at the center of each quadrant, with the text size automatically adapting to fit.

At the beginning, the `setup()` function creates a full-window canvas using `createCanvas(windowWidth, windowHeight)`. It sets rectangles to be drawn from corner to corner with `rectMode(CORNERS)`, centers all text via `textAlign(CENTER, CENTER)`, and hides the default mouse cursor to display a custom indicator later.

Inside the main drawing loop, `draw()`, the program continuously updates the scene:

- It defines the coordinates of the four corners of the canvas (top-left, top-right, bottom-left, bottom-right), and uses `constrain()` to keep the mouse position within window bounds.
- It calculates the **base and height** of each rectangle, and thus its area (e.g., the top-left quadrant's area is `areaTL = abs(mx) * abs(my)`).
- Using `min()` and `max()`, it determines the smallest and largest areas among the four quadrants, mapping each area to a grayscale value using `map()`. The smallest area is black, the largest is white, and intermediate values are mapped to various shades of gray.
- Each area is converted from pixels squared to square centimeters with the formula `area_cm2 = area_px / (37.8 * 37.8)`, where 37.8 is an approximate number of pixels per centimeter on a standard display.
- The rectangles are drawn using the computed grayscale value with `fill()`, and outlined in thick black (`stroke(0)` and `strokeWeight(6)`).
- At the center of each rectangle, the program displays the area value (in cm²) using `text()`. The text color is chosen automatically (white or black) for visibility based on the rectangle's background, and the text size dynamically adjusts so that it never overflows the quadrant.
- Finally, a small yellow circle with a black border is drawn at the current mouse position, acting as a clear and modern custom pointer.

This system allows for real-time visualization of how the canvas is subdivided and how the area of each rectangle changes as you move the mouse, providing both a graphical (color) and numerical (area) feedback for each quadrant

```
function setup() {
  createCanvas(windowWidth, windowHeight); // Full window canvas
  rectMode(CORNERS);                       // Draw rectangles using corner points
  textAlign(CENTER, CENTER); // Center text within rectangles
  noCursor();                // Hide mouse cursor
}

function draw() {
  background(220); // Light gray background

  // Define canvas corners
  let tlX = 0, tlY = 0;
  let trX = width, trY = 0;
  let blX = 0, blY = height;
  let brX = width, brY = height;

  // Clamp mouse position within canvas
  let mx = constrain(mouseX, 0, width);
  let my = constrain(mouseY, 0, height);

  // Calculate each quadrant area in pixels
  let areaTL = abs(mx) * abs(my);
  let areaTR = abs(width - mx) * abs(my);
  let areaBL = abs(mx) * abs(height - my);
  let areaBR = abs(width - mx) * abs(height - my);

  // Get min and max area for grayscale mapping
  let areaMin = min(areaTL, areaTR, areaBL, areaBR);
  let areaMax = max(areaTL, areaTR, areaBL, areaBR);

  // Map area to grayscale (darker = smaller area)
  let grayTL = map(areaTL, areaMin, areaMax, 0, 255);
  let grayTR = map(areaTR, areaMin, areaMax, 0, 255);
  let grayBL = map(areaBL, areaMin, areaMax, 0, 255);
  let grayBR = map(areaBR, areaMin, areaMax, 0, 255);

  stroke(0);
  strokeWeight(6); // Thick black outlines (Mondrian-like)

  // Convert pixel areas to cm² for display
  let px_per_cm = 37.8;
  let areaTL_cm2 = areaTL / (px_per_cm * px_per_cm);
  let areaTR_cm2 = areaTR / (px_per_cm * px_per_cm);
  let areaBL_cm2 = areaBL / (px_per_cm * px_per_cm);
  let areaBR_cm2 = areaBR / (px_per_cm * px_per_cm);

  // Draw rectangles with dynamic grayscale fill
  fill(grayTL); rect(tlX, tlY, mx, my);
  fill(grayTR); rect(trX, trY, mx, my);
  fill(grayBL); rect(blX, blY, mx, my);
  fill(grayBR); rect(brX, brY, mx, my);

  // --- Adaptive text size for each rectangle ---
  let maxTextSize = 26;
  let padding = 10;

  // Top-left quadrant
  let wTL = abs(mx), hTL = abs(my);
  let tsTL = min(maxTextSize, max(10, 0.5 * min(wTL, hTL) - padding));
  textSize(tsTL);
  fill(grayTL < 128 ? 255 : 0); // White text on dark, black on light
  noStroke();
  text(nf(areaTL_cm2, 1, 2) + " cm²", mx / 2, my / 2);

  // Top-right quadrant
  let wTR = abs(width - mx), hTR = abs(my);
  let tsTR = min(maxTextSize, max(10, 0.5 * min(wTR, hTR) - padding));
  textSize(tsTR);
  fill(grayTR < 128 ? 255 : 0);
  text(nf(areaTR_cm2, 1, 2) + " cm²", mx + wTR / 2, my / 2);

  // Bottom-left quadrant
  let wBL = abs(mx), hBL = abs(height - my);
  let tsBL = min(maxTextSize, max(10, 0.5 * min(wBL, hBL) - padding));
  textSize(tsBL);
  fill(grayBL < 128 ? 255 : 0);
  text(nf(areaBL_cm2, 1, 2) + " cm²", mx / 2, my + hBL / 2);

  // Bottom-right quadrant
  let wBR = abs(width - mx), hBR = abs(height - my);
  let tsBR = min(maxTextSize, max(10, 0.5 * min(wBR, hBR) - padding));
  textSize(tsBR);
  fill(grayBR < 128 ? 255 : 0);
  text(nf(areaBR_cm2, 1, 2) + " cm²", mx + wBR / 2, my + hBR / 2);

  textSize(maxTextSize); // Restore default text size

  // Draw yellow circle at mouse position (with black outline)
  stroke(0);
  strokeWeight(3);
  fill(255, 220, 30);
  ellipse(mx, my, 18, 18);
}

// END

/*
  This interactive sketch divides the canvas into four rectangles, updating dynamically with the mouse.
  Each rectangle's area and shade reflect its size; adaptive text displays each area in cm².
  The Mondrian-style outlines and a yellow marker highlight the real-time geometry as the mouse moves.
*/
```