Subtraction Strategies: Counting On/Back By Bases and then Ones (CBBO)

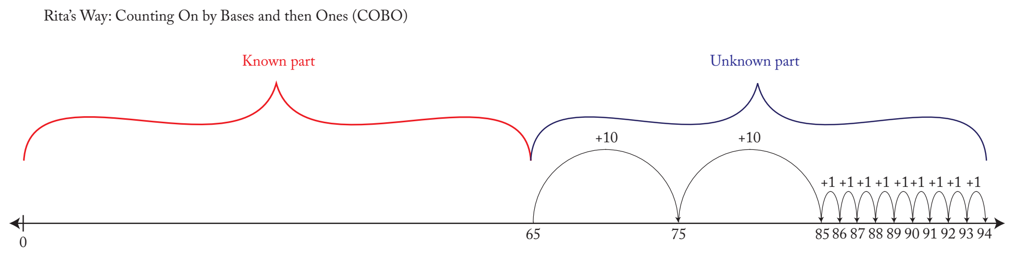
Compiled by: Theodore M. Savich

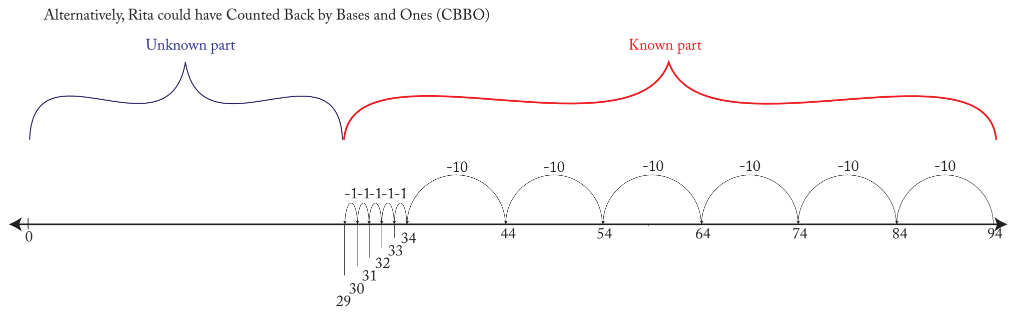
March 28, 2025

Transcript

Video from Carpenter et al. (1999). Strategy descriptions and examples adapted from Hackenberg (2025).

- **Teacher:** Earl had a collection of 65 bird feathers, on a trip to a marsh he found lots more feathers to put in his collection. Now he has 94 feathers in his collection. How many feathers did Earl find at the marsh?
- **Rita** So he had what?
- **Teacher:** He started off with, 65 feathers.
- **Rita:** 1,2,3,4,5,6 1,2,3,4,5. And then he had how many?
- **Teacher:** Well, he had 65 bird feathers. On a trip to a marsh, he found lots more and he put them in his collection. Now he has 94.
- **Rita:** Well, I can 65, 75, 85. How many did he find?
- **Teacher:** Well, that's my question for you. How many did he find? He ends up with 94.
- **Rita:** And 85,86,87,88,89,90, 91,92,93,94 and so the answer is 20, 21, 22, 23, 24, 25, 26, 27, 28, 29.
- **Teacher** Nice work.





Notation Representing Rita's Solution:

$$\begin{aligned} 65 + (10) &= 75 \\ 75 + (10) &= 85 \\ 85 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 &= 94 \\ 10 + 10 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 &= 29 \end{aligned}$$

Description of Strategy:

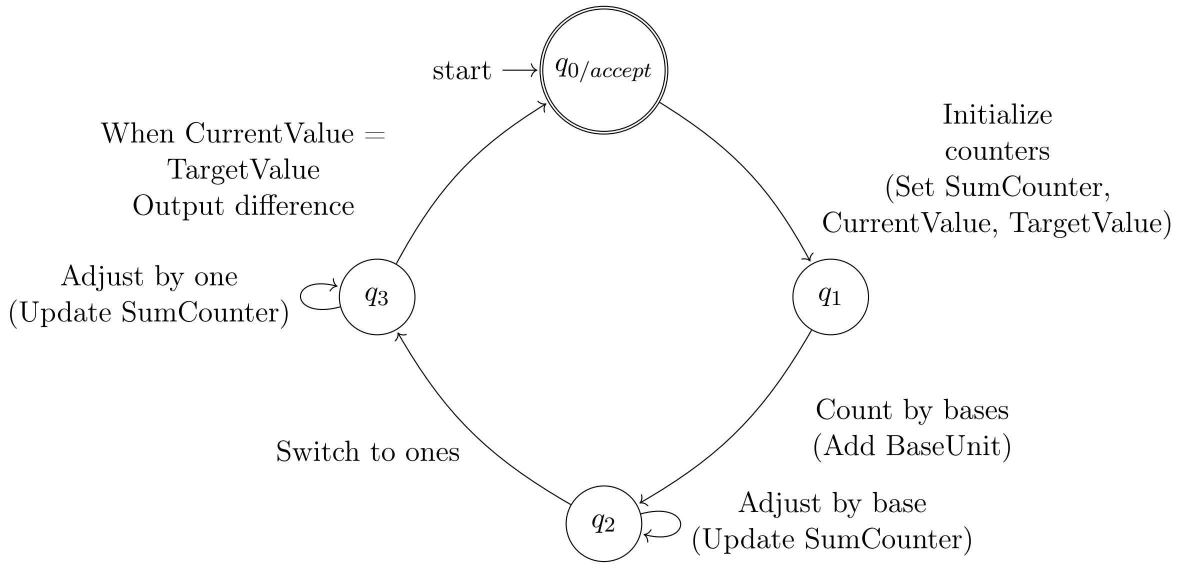
Objective: Description of Counting On by Bases and Then Ones (COBO) Begin with one of the numbers. Break the other number into its base units and its ones. Then, "count on" by adding each base unit one at a time, followed by each individual one.

Why are number lines useful for demonstrating this strategy? COBO is essentially a jump strategy—you start at one number and make "jumps" equal to the other number's base units, then add in the remaining ones. Number lines are ideal because they visually display jumps of varying lengths and directions. They serve as a picture of the process: a jump representing a full base is clearly larger (by a factor of the base) than a jump of a single unit.

Good number line illustrations should:

- Clearly represent the relative sizes of the jumps—each base jump should be exactly as many times larger than a single-unit jump as the base indicates, with all base jumps the same size and all one-unit jumps identical.
- Indicate the position of 0, or mark a break if that portion of the line isn't drawn to scale.
- Use arrows to indicate direction—when adding, the jumps go to the right (or upward); when subtracting, they go to the left (or downward).
- Mark all landing points clearly—the numbers you would speak aloud when counting on by bases and then ones, just as Lauren demonstrated.

Automaton Diagram for Counting On or Back by Bases and Then Ones


start
$q_{0/accept}$
Initialize counters (Set SumCounter, CurrentValue, TargetValue)
q_1
Count by bases (Add BaseUnit)
q_2
Adjust by base (Update SumCounter)
Switch to ones
q_3
Adjust by one (Update SumCounter)
When CurrentValue = TargetValue Output difference

HTML Implementation

```
<!DOCTYPE html>
<html>
<head>
    <title>Subtraction Strategies: Counting Back By Bases and Ones (CBBO)</title>
    <style>
        body { font-family: sans-serif; }
        #diagramCBBOSVG { border: 1px solid #d3d3d3; }
        #outputContainer { margin-top: 20px; }
        .number-line-tick { stroke: black; stroke-width: 1; }
        .number-line-break { stroke: black; stroke-width: 1; } /* Solid for zig-zag */
        .number-line-label { font-size: 12px; text-anchor: middle; }
        .jump-arrow { fill: none; stroke: purple; stroke-width: 1.5; } /* CBBO color */
        .jump-arrow-head { fill: purple; stroke: purple; } /* CBBO color */
        .jump-label { font-size: 10px; text-anchor: middle; fill: purple; } /* CBBO color
            */
        .tens-jump-label { font-size: 12px; text-anchor: middle; fill: purple; }
        .stopping-point { fill: red; stroke: black; stroke-width: 1; }
        .number-line-arrow { fill: black; stroke: black; }
        .extended-tick { stroke: black; stroke-width: 1; } /* Reuse COBO style */
    </style>
</head>
<body>

<h1>Subtraction Strategies: Counting Back By Bases and Then Ones (CBBO)</h1>

<div>
    <label for="cbboMinuend">Minuend:</label>
    <input type="number" id="cbboMinuend" value="94"> <!-- Example from PDF -->
</div>
<div>
    <label for="cbboSubtrahend">Subtrahend:</label>
    <input type="number" id="cbboSubtrahend" value="29"> <!-- 94 - 65 = 29 -->
</div>

<button onclick="runCBBOAutomaton()">Calculate and Visualize</button>

<div id="outputContainer">
    <h2>Explanation:</h2>
    <div id="cbboOutput">
        <!-- Text output will be displayed here -->
    </div>
</div>

<h2>Diagram:</h2>
<svg id="diagramCBBOSVG" width="700" height="350"></svg>

<script>
document.addEventListener('DOMContentLoaded', function() {
    const outputElement = document.getElementById('cbboOutput');
    const cbboMinuendInput = document.getElementById('cbboMinuend');
    const cbboSubtrahendInput = document.getElementById('cbboSubtrahend');
    const diagramCBBOSVG = document.getElementById('diagramCBBOSVG');
```

```

// --- Helper Functions (Keep createText, drawTick, drawScaleBreakSymbol,
    createJumpArrow, drawStoppingPoint from previous corrected versions) ---
function createText(svg, x, y, textContent, className = 'number-line-label') {
    const text = document.createElementNS("http://www.w3.org/2000/svg", 'text');
    text.setAttribute('x', x);
    text.setAttribute('y', y);
    text.setAttribute('class', className);
    text.setAttribute('text-anchor', 'middle'); // Center labels
    text.textContent = textContent;
    svg.appendChild(text);
}

function drawTick(svg, x, y, size) {
    const tick = document.createElementNS('http://www.w3.org/2000/svg', 'line');
    tick.setAttribute('x1', x);
    tick.setAttribute('y1', y - size / 2);
    tick.setAttribute('x2', x);
    tick.setAttribute('y2', y + size / 2);
    tick.setAttribute('class', 'number-line-tick');
    svg.appendChild(tick);
}

 function drawScaleBreakSymbol(svg, x, y) {
    const breakOffset = 4;
    const breakHeight = 8;
    const breakLine1 = document.createElementNS('http://www.w3.org/2000/svg', 'line');
    breakLine1.setAttribute('x1', x - breakOffset);
    breakLine1.setAttribute('y1', y - breakHeight);
    breakLine1.setAttribute('x2', x + breakOffset);
    breakLine1.setAttribute('y2', y + breakHeight);
    breakLine1.setAttribute('class', 'number-line-break');
    svg.appendChild(breakLine1);
    const breakLine2 = document.createElementNS('http://www.w3.org/2000/svg', 'line');
    breakLine2.setAttribute('x1', x + breakOffset);
    breakLine2.setAttribute('y1', y - breakHeight);
    breakLine2.setAttribute('x2', x - breakOffset);
    breakLine2.setAttribute('y2', y + breakHeight);
    breakLine2.setAttribute('class', 'number-line-break');
    svg.appendChild(breakLine2);
}

 function createJumpArrow(svg, x1, y1, x2, y2, jumpArcHeight, direction = 'forward',
     arrowSize = 5) { // Removed default color, use CSS
     const path = document.createElementNS('http://www.w3.org/2000/svg', 'path');
     const cx = (x1 + x2) / 2;
     const cy = y1 - jumpArcHeight; // Arc is above the line
     path.setAttribute('d', `M ${x1} ${y1} Q ${cx} ${cy} ${x2} ${y1}`); // Use y1 for
         end point to land on line
     path.setAttribute('class', `jump-arrow`); // Rely on CSS for color
     svg.appendChild(path);

     // Arrowhead
     const arrowHead = document.createElementNS('http://www.w3.org/2000/svg', 'path');
```

```
        const dx = x2 - cx;
        const dy = y1 - cy; // Use y1 for angle calculation
        const angleRad = Math.atan2(dy, dx);
        let angleDeg = angleRad * (180 / Math.PI);
        arrowHead.setAttribute('class', `jump-arrow-head`); // Rely on CSS for color

        if (direction === 'forward') {
            angleDeg += 180; // Point right
            arrowHead.setAttribute('d', `M 0 0 L ${arrowSize} ${arrowSize/2} L ${
                arrowSize} ${-arrowSize/2} Z`);
            arrowHead.setAttribute('transform', `translate(${x2}, ${y1}) rotate(${
                angleDeg})`);
        } else { // backward
            // angleDeg already points left-ish from Q curve end
            arrowHead.setAttribute('d', `M 0 0 L ${-arrowSize} ${arrowSize/2} L ${-
                arrowSize} ${-arrowSize/2} Z`); // Pointy part is at (0,0)
             // We want to rotate to align with the curve's end direction
            arrowHead.setAttribute('transform', `translate(${x2}, ${y1}) rotate(${
                angleDeg})`);
        }
        svg.appendChild(arrowHead);
    }

    function drawStoppingPoint(svg, x, y, labelText, labelOffsetBase) {
        const circle = document.createElementNS('http://www.w3.org/2000/svg', 'circle');
        circle.setAttribute('cx', x);
        circle.setAttribute('cy', y);
        circle.setAttribute('r', 5);
        circle.setAttribute('class', 'stopping-point');
        svg.appendChild(circle);
        createText(svg, x, y + labelOffsetBase * 1.5, labelText, 'number-line-label');
    }
   // --- End Helper Functions ---

   // --- Main CBBO Automaton Function ---
   window.runCBBOAutomaton = function() {
       try {
           const minuend = parseInt(cbboMinuendInput.value);
           const subtrahend = parseInt(cbboSubtrahendInput.value); // Amount to subtract
           if (isNaN(minuend) || isNaN(subtrahend)) {
               outputElement.textContent = 'Please enter valid numbers for Minuend and 
                   Subtrahend';
               diagramCBBOSVG.innerHTML = '';
               return;
           }
            if (subtrahend > minuend) {
               outputElement.textContent = 'Subtrahend cannot be greater than Minuend for 
                   CBBO.';
               diagramCBBOSVG.innerHTML = '';
               return;
           }

           let output = `<h2>Counting Back by Bases and Ones (CBBO)</h2>\n\n`;
           output += `<p><strong>Problem:</strong> ${minuend} - ${subtrahend}</p>\n\n`;

           const tensToSubtract = Math.floor(subtrahend / 10) * 10;
           const onesToSubtract = subtrahend % 10;

           output += `Step 1: Split subtrahend ${subtrahend} into ${tensToSubtract} + ${onesToSubtract}\n\n`;

           let currentVal = minuend;
           const tensSteps = [];
           if (tensToSubtract > 0) {
               output += 'Step␣2:␣Count␣back␣by␣tens\n';
               for (let i = 10; i <= tensToSubtract; i += 10) {
                   tensSteps.push({ from: currentVal, to: currentVal - 10, action: 'Subtract␣10' });
                   currentVal -= 10;
               }
               tensSteps.forEach(step => {
                   output += `<p>${step.from} - 10 = ${step.to}</p>\n`; // Simplified text
               });
               output += '\n';
           }

           const onesSteps = [];
           if (onesToSubtract > 0) {
               output += `Step ${tensToSubtract > 0 ? '3' : '2'}: Count back by ones\n`;
               for (let i = 1; i <= onesToSubtract; i++) {
                   onesSteps.push({ from: currentVal, to: currentVal - 1, action: 'Subtract␣1' });
                   currentVal -= 1;
               }
               onesSteps.forEach(step => {
                   output += `<p>${step.from} - 1 = ${step.to}</p>\n`; // Simplified text
               });
               output += '\n';
           }

           const finalDifference = currentVal; // The final landing spot IS the difference
           output += `Result: ${minuend} - ${subtrahend} = ${finalDifference}`;
           outputElement.innerHTML = output;
           typesetMath();

           // Draw the diagram
           drawCBBONumberLineDiagram(diagramCBBOSVG, minuend, subtrahend, tensSteps, onesSteps, finalDifference);


       } catch (error) {
            console.error("Error␣in␣runCBBOAutomaton:", error);
           outputElement.textContent = `Error: ${error.message}`;
       }
   };

```

```
function drawCBBONumberLineDiagram(svg, minuend, subtrahend, tensSteps, onesSteps,
    finalDifference) {
    if (!svg || typeof svg.setAttribute !== 'function') { return; }
    svg.innerHTML = '';

    const svgWidth = parseFloat(svg.getAttribute('width'));
    const svgHeight = parseFloat(svg.getAttribute('height'));
    const startX = 50;
    const endX = svgWidth - 50;
    const numberLineY = svgHeight / 2; // Center vertically
    const tickHeight = 10;
    const labelOffsetBase = 20;
    const jumpHeight = 30; // Consistent jump height for CBBO
    const jumpLabelOffset = 15;
    const arrowSize = 5;
    const scaleBreakThreshold = 40;

    // Determine range for scaling
    let diagramMin = finalDifference;
    let diagramMax = minuend;

    // Calculate scale and handle potential break (near 0, before diagramMin)
    let displayRangeStart = diagramMin;
    let scaleStartX = startX;
    let drawScaleBreak = false;

    if (diagramMin > scaleBreakThreshold) {
        displayRangeStart = diagramMin - 10;
        scaleStartX = startX + 30;
        drawScaleBreak = true;
        drawScaleBreakSymbol(svg, scaleStartX - 15, numberLineY);
        drawTick(svg, startX, numberLineY, tickHeight);
        createText(svg, startX, numberLineY + labelOffsetBase, '0', 'number-line-label
            ');
    } else {
        displayRangeStart = 0;
        drawTick(svg, startX, numberLineY, tickHeight);
        createText(svg, startX, numberLineY + labelOffsetBase, '0', 'number-line-label
            ');
    }

    const displayRangeEnd = diagramMax + 10;
    const displayRange = Math.max(displayRangeEnd - displayRangeStart, 1);
    const scale = (endX - scaleStartX) / displayRange;

    // Function to convert value to X coordinate
    function valueToX(value) {
        if (value < displayRangeStart && drawScaleBreak) { return scaleStartX - 10; }
        const scaledValue = scaleStartX + (value - displayRangeStart) * scale;
        return Math.max(scaleStartX, Math.min(scaledValue, endX));
    }

    // Draw the main visible segment of the number line
     const mainLineStartX = valueToX(displayRangeStart);
```

```
         const mainLineEndX = valueToX(displayRangeEnd);
         const numberLine = document.createElementNS(’http://www.w3.org/2000/svg’, ’line’);
         numberLine.setAttribute(’x1’, mainLineStartX);
         numberLine.setAttribute(’y1’, numberLineY);
         numberLine.setAttribute(’x2’, mainLineEndX);
         numberLine.setAttribute(’y2’, numberLineY);
         numberLine.setAttribute(’class’, ’number-line-tick’);
         svg.appendChild(numberLine);

         // Add arrowhead to the right end
         const mainArrowHead = document.createElementNS(’http://www.w3.org/2000/svg’, ’path’);
         mainArrowHead.setAttribute(’d’, ‘M ${mainLineEndX - arrowSize} ${numberLineY - arrowSize/2} L ${mainLineEndX} ${numberLineY} L ${mainLineEndX - arrowSize} ${numberLineY + arrowSize/2} Z‘);
         mainArrowHead.setAttribute(’class’, ’number-line-arrow’);
         svg.appendChild(mainArrowHead);


        // Draw Ticks and Labels
        function drawTickAndLabel(value, index) {
            const x = valueToX(value);
            if (x < scaleStartX - 5 && value !== 0) return;

            drawTick(svg, x, numberLineY, tickHeight);
            const labelOffset = labelOffsetBase * (index % 2 === 0 ? 1 : -1.5); // Stagger
            createText(svg, x, numberLineY + labelOffset, value.toString(), ’number-line-label’);
        }

        // Collect all points to draw ticks for
        let allPoints = new Set([minuend, finalDifference]); // Start and end
        tensSteps.forEach(step => allPoints.add(step.to));
        onesSteps.forEach(step => allPoints.add(step.to));
        let sortedPoints = Array.from(allPoints).sort((a, b) => a - b);
        let pointIndexMap = {};
        let currentIndex = 0;
        sortedPoints.forEach(point => {
            if (point >= displayRangeStart || (point === 0 && !drawScaleBreak)) {
                if (!(point < displayRangeStart && drawScaleBreak)) {
                    pointIndexMap[point] = currentIndex++;
                    drawTickAndLabel(point, pointIndexMap[point]);
                }
            }
        });

        // Draw tens jumps (Backward)
        tensSteps.forEach((step, index) => {
            const x1 = valueToX(step.from);
            const x2 = valueToX(step.to);
            if (x1 <= scaleStartX || x2 < scaleStartX) return; // Skip if outside visible range

            const staggerOffset = index % 2 === 0 ? 0 : jumpHeight * 0.5;
            createJumpArrow(svg, x1, numberLineY, x2, numberLineY, jumpHeight + staggerOffset, 'backward', arrowSize);
            createText(svg, (x1 + x2) / 2, numberLineY - (jumpHeight + staggerOffset) - jumpLabelOffset, '-10', 'tens-jump-label');
        });

        // Draw ones jumps (Backward)
        onesSteps.forEach((step, index) => {
            const x1 = valueToX(step.from);
            const x2 = valueToX(step.to);
             if (x1 <= scaleStartX || x2 < scaleStartX) return; // Skip if outside visible range

            const staggerOffset = (tensSteps.length + index) % 2 === 0 ? 0 : jumpHeight * 0.5; // Continue staggering
            createJumpArrow(svg, x1, numberLineY, x2, numberLineY, jumpHeight + staggerOffset, 'backward', arrowSize);
            createText(svg, (x1 + x2) / 2, numberLineY - (jumpHeight + staggerOffset) - jumpLabelOffset, '-1', 'jump-label');
        });

        // Start point marker
        if (valueToX(minuend) >= scaleStartX) {
            drawStoppingPoint(svg, valueToX(minuend), numberLineY, 'Start', labelOffsetBase);
        }
    }

    function typesetMath() { /* Placeholder */ }

    // Initial run on page load
    runCBBOAutomaton();

});
</script>

</body>
</html>
```

References

Carpenter, T. P., Fennema, E., Franke, M. L., Levi, L., & Empson, S. B. (1999). Children's mathematics: Cognitively guided instruction – videotape logs [supplementary material]. In *Children's mathematics: Cognitively guided instruction*. Heinemann, in association with The National Council of Teachers of Mathematics, Inc.

Hackenberg, A. (2025). *Course notes* [Unpublished course notes].